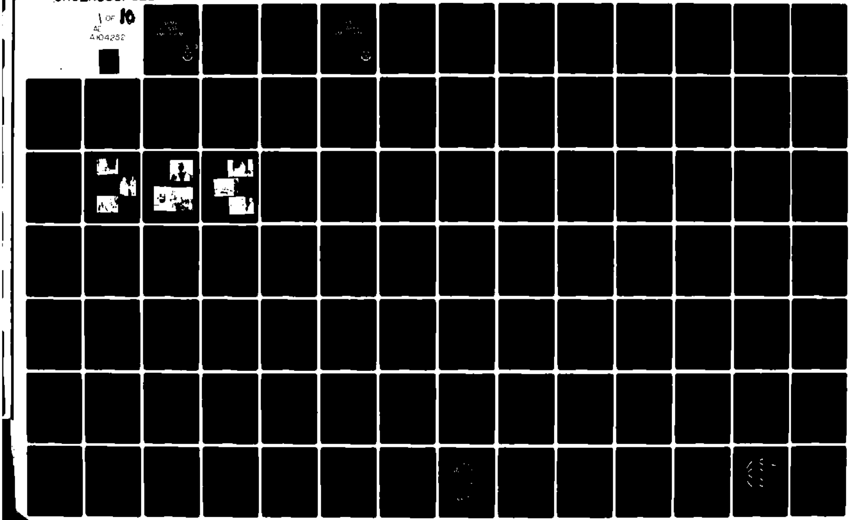
AD-A104 252 INTERNATIONAL BUSINESS SERVICES INC WASHINGTON DC F/G 9/2
US ARMY SOFTWARE SYMPOSIUM (2ND) HELD AT WILLIAMSBURG, VIRGINIA--ETC(U)
1978 S M TAYLOR DAAK70-78-D-0030

UNCLASSIFIED NL

1 OF 10
AD
A104252



SECOND

ADA104252

U.S. ARMY SOFTWARE SYMPOSIUM

OCTOBER 25-27, 1978
WILLIAMSBURG, VIRGINIA

SPONSORED BY
THE U.S. ARMY
COMPUTER SYSTEMS
COMMAND

COMPUTER SYSTEMS COMMAND
UNITED STATES ARMY

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(12) 875

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

(6)

2. GOVT ACCESSION NO.

AD-A104252

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

Second US Army Software Symposium (2nd) Held at Williamsburg, Virginia on 25-27 Oct

5. TYPE OF REPORT & PERIOD COVERED

Final - 25-27 October 1978

6. PERFORMING ORG. REPORT NUMBER

1178.

7. AUTHOR(s)

(10) Stanley M. Taylor

8. CONTRACT OR GRANT NUMBER(s)

(15) DAAK70-78-D-0030

9. PERFORMING ORGANIZATION NAME AND ADDRESS

International Business Services, Inc.
1090 Vermont Avenue, NW Suite 1010
Washington, D.C. 20005

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

(11) 1978

11. CONTROLLING OFFICE NAME AND ADDRESS

US Army Institute for Research in Management
Information and Computer Science (AIRMICS),
115 O'Keefe Building, GIT, Atlanta, GA 30332

12. REPORT DATE

25-27 October 1978

13. NUMBER OF PAGES

854

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)

(9) Final rept,

15. SECURITY CLASS. (of this report)

UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

DISTRIBUTION [redacted] A

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Functional System Requirements, Human Factors, Life Cycle Management, Security, Softwwre Engineering Tools, Testbeds, Interoperability, Tradeoffs, Graphics, Verification, Standardization/Commonality, Survivability.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The following sessions were conducted during this symposium:

Functional Systems Requirements

Human Factors

Requirements I

Life Cycle Management

UNCLASSIFIED 343157
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Security

Software Engineering Tools & Methods I

Requirements II

Patriot Software System

Testbeds

Software Engineering Tools & Methods II

Interoperability

Management Control Technology

Hardware/Firmware/Software Tradeoffs

Graphics

Formal Methods of Software Verification and Maintenance

Auto Test/Diagnostic Equipment & Software

Computer Architecture Standardization/Commonality

Reliability/Survivability

SECOND

U.S. ARMY SOFTWARE SYMPOSIUM

OCTOBER 25-27, 1978
WILLIAMSBURG, VIRGINIA

SPONSORED BY
THE U.S. ARMY
COMPUTER SYSTEMS
COMMAND

COMPUTER SYSTEMS COMMAND
UNITED STATES ARMY

SECOND U.S. ARMY SOFTWARE SYMPOSIUM

Williamsburg, Virginia

25-27 October, 1978

SPONSOR

UNITED STATES ARMY COMPUTER SYSTEMS COMMAND
(USACSC)

SYMPOSIUM CHAIRMAN

Dr. Stanley M. Taylor
Aberdeen Proving Ground

SYMPOSIUM DIRECTOR

R. Peyton Brown - USACSC
Integrated Software Research & Development Working Group

DEPUTY SYMPOSIUM CHAIRPERSON

Dr. John Staudhammer
Army Research Institute

SYMPOSIUM PLANNING COMMITTEE

Merton J. Batchelder
U.S. Army Computer Systems Command

LTC Roy Busdiecker
Office of Assist. Chief of Staff for Automation & Communication

John M. Cole
Center for Tactical Computer Sciences

Allan H. Curry
U.S. Army Institute in Management Information & Computer Sciences

Jean N. Hooper
Army Research Institute

James M. Jones II
U.S. Army Engineers Waterways Experiment Station

Stephen P. Kroll
Deputy Chief of Staff for Research Development and Acquisition

Sil Pelosi
Center for Tactical Ccmputer Sciences

Dr. N. Radhadrishnan
U.S. Army Engineers Waterways Experiment Station

Robert Rosen
Harry Diamond Labs

Norman J. Taupeka
Center for Tactical Computer Sciences

Proceedings Prepared/Delivered By
International Business Services, Inc.

Under Contract DAAK 70-78-D-0030

TABLE OF CONTENTS

TABLE OF CONTENTS Cont'd

LIFE CYCLE MANAGEMENT

Chairman: MAJ Edward H. Ely

SECURITY

Chairman: LTC Robert P. Campbell

SOFTWARE ENGINEERING TOOLS & METHODS I

Chairman: Derek S. Morris

TABLE OF CONTENTS Cont'd

TABLE OF CONTENTS Cont'd

TITLE PAGE

SOFTWARE ENGINEERING TOOLS & METHODS II

Chairman: Dr. Mehdi Jazayeri

INTEROPERABILITY

Chairman: Dwaine B. Huewe

MANAGEMENT CONTROL TECHNOLOGY

Chairman: Allan Curry

HARDWARE/FIRMWARE/SOFTWARE TRADEOFFS

Chairman: Dr. Serafino Amoroso

TABLE OF CONTENTS Cont'd

TITLE PAGE

GRAPHICS

Chairman: Dr. N. Radhakrishnan

FORMAL METHODS OF SOFTWARE VERIFICATION AND MAINTENANCE

Chairman: Norman J. Taupeka

TABLE OF CONTENTS Cont'd

THE SECOND U.S. ARMY SOFTWARE SYMPOSIUM

October 25-27, 1978
Williamsburg, Virginia

AGENDA

TIME	EVENT	PLACE
	<u>WEDNESDAY, OCTOBER 25</u>	
Noon-6:00 PM	Registration	West Gallery
1:30-5:00 PM	Microprocessing Workshop Dr. James Gault Dr. Wesley Snyder North Carolina State University	Jamestown
	<u>THURSDAY, OCTOBER 26</u>	
8:00-11:00 AM	Registration	West Gallery
8:30-10:00 AM	OVERVIEW Welcome Address BG Leonard J. Riley INTRODUCTIONS Dr. Stanley Taylor Symposium Chairperson *The Defense System Software Management Program-- A Balance Between Management and Technology* KEYNOTE SPEAKER: Barry C. DeRoze Manager of Advanced Systems TRW Defense and Space System Group	Jamestown

TIME	EVENT	PLACE
	Strategy for Management *of* *Defense System Computer Resources*	
	KEYNOTE SPEAKER: H. Mark Grove Acting Assistant for Defense System Software Office of Deputy Undersecretary of Defense (Acquisition Policy) Directorate of Materiel Acquisition Policy	
10:00-10:30 AM	BREAK	
10:30 AM-Noon	PLENARY SESSIONS	
	● FUNCTIONAL SYSTEMS REQUIREMENTS	Yorktown
	SESSION CHAIRPERSON: COL. Ray Ketchum TRADOC	
	Army Battlefield Automation Architecture Major James H. Helberg Dennis P. Mahoney USACACDA, Fort Leavenworth	
	Army Battlefield Interface Concept Major M. W. Robinson Captain John T. Ratzenberger USACACDA, Fort Leavenworth	
	Techniques for Controlling Proliferation *of* *Automation on the Battlefield* Major Richard D. James Dr. Edward R. Fowler USACACDA, Fort Leavenworth	
	● HUMAN FACTORS	Jamestown
	SESSION CHAIRPERSON: Jean N. Hooper Army Research Institute	
	Human Performance in Software Development Jean N. Hooper Army Research Institute	
	Human Factors in Query Language Lawrence M. Potash Army Research Institute	

THE SECOND U.S. ARMY SOFTWARE SYMPOSIUM

October 25-27, 1978
Williamsburg, Virginia

AGENDA

TIME	EVENT	PLACE
	WEDNESDAY, OCTOBER 25	
Noon-6:00 PM	Registration	West Gallery
1:30-5:00 PM	Microprocessing Workshop Dr. James Gault Dr. Wesley Snyder North Carolina State University	Jamestown
	THURSDAY, OCTOBER 26	
8:00-11:00 AM	Registration	West Gallery
8:30-10:00 AM	OVERVIEW Welcome Address BG Leonard J. Riley INTRODUCTIONS Dr. Stanley Taylor Symposium Chairperson *The Defense System Software Management Program-- A Balance Between Management and Technology* KEYNOTE SPEAKER: Barry C. DeRoze Manager of Advanced Systems TRW Defense and Space System Group	Jamestown

TIME	EVENT	PLACE
	Issues in Human - Computer Interaction Raymond C. Sidorsky Army Research Institute	
Noon-1:30 PM	LUNCH Deli Buffet	Main Dining Room
1:30-3:00 PM	PLENARY SESSIONS	Locations to be listed in West Gallery
	● REQUIREMENTS I SESSION CHAIRPERSON: Dr. Edward Lieblein CENTACS	
	Computer Aided Requirements Generation - An Evaluation Carl G. Davis BMD Advanced Technology Center	
	An Approach to Requirements Definition For Real-Time System David Egli U.S. Army Communications Research and Development Command Center for Tactical Computer Systems Software Engineering Division Fort Monmouth	
	Nuts and Bolts of Software Acquisition Thomas A. Rorro U.S. Army Electronics Research and Development Command Beta Joint Project Office Integration Division	
	● LIFE CYCLE MANAGEMENT SESSION CHAIRPERSON: Edward H. Ely AIRMICS	
	Toward Understanding the Software Life Cycle Victor R. Basili Department of Computer Science University of Maryland	
	Modeling's Role in Determining Software System Complexity Thomas G. DeLutis, Ph.D Department of Computer and Information Science The Ohio State University	

TIME	EVENT	PLACE
	Contingency Theory Approach *To* *Systems Life Cycle Management* J. David Naumann & Gordon B. Davis University of Minnesota	
	● OPERATING SYSTEMS SECURITY SESSION CHAIRPERSON: LTC. Robert P. Campbell DAMI-AM	
	The Department of Defense *Kernelized Secure Operating System* *(KSOS)* E. J. McCauley Ford Aerospace and Communications Corporation	
	● SOFTWARE ENGINEERING TOOLS & METHODS I SESSION CHAIRPERSON: Derek S. Morris	
	The Department of Defense *Common Programming Language Project* Serafino Amoroso CENTACS Fort Monmouth	
	An Experimental Application *of* *The DOD Common Language* *to* *A Telecommunications System Design* Derek S. Morris CENTACS Fort Monmouth	
	An Integrated System of Tools *to Support the DOD Common Language* Dennis J. Turner CENTACS Fort Monmouth	
3:00-3:30 PM	BREAK	
3:00-5:00 PM	PLENARY SESSIONS CONTINUED	

TIME EVENT PLACE

● REQUIREMENTS II

SESSION CHAIRPERSON: John Mitchell
AIRMICS

*Developments and Problems
in
Software Requirements*

Pei Hsia &
Bill Buckles

*User Experience
with a
Formally Defined Requirements Language
(IORL)*

Charles R. Everhart
Teledyne Brown Engineering

*Software Requirements Engineering Methodology
(SREM)*

M. W. Alford
TRW Defense and Space Systems Group

Automated Analysis of System Specifications

Dr. Larry A. Johnson
Dr. Paul B. Merrithew
Mr. Daniel G. Smith
LOGICON, Inc.

● PATRIOT SOFTWARE SYSTEM

Edward U. Lee, Manager
Patriot Software Development
Missile Systems Division
Rayethon Company

● TEST BEDS

SESSION CHAIRPERSON: John M. Cole
System Validation Division
CENTACS

*Emulation of Tactical Data Systems
in the
Teleprocessing Design Center*

John M. Cole
CENTACS

Interfacing CS^3 *Facilities
to the Arpanet*

Marvin Schwartz
CENTACS

TIME	EVENT	PLACE
	Software Development Support System *(SDSS)*	
	Bernard Newman Roy Mattson CENTACS	
	● SOFTWARE ENGINEERING: TOOLS AND METHODS II	
	SESSION CHAIRPERSON: Dr. M. Jazayeri AIRMICS and The University of North Carolina	
	Current Developments in Program Verification	
	Susan L. Gerhart Information Sciences Institute University of Southern California	
	The Human Engineering *of a* *System Design Environment* *for* *Microprocessor-Based Systems*	
	Charles W. Rose Donald C. Hewitt, Jr. Case Western Reserve Univeristy	
6:00-7:00 PM	ATTITUDE ADJUSTMENT HOUR	
7:00-9:00 PM	BANQUET	Jamestown
	● INTRODUCTION	
	Dr. John Staudhammer Deputy Symposium Chairperson	
	The Challenge of Software Development	
	BANQUET SPEAKER: MG Clay T. Buckingham	
	FRIDAY, OCTOBER 27	
8:30-10:00 AM	PLENARY SESSIONS	Locations to be listed in West Gallery
	● INTEROPERABILITY SESSION	
	SESSION CHAIRPERSON: Dwaine B. Huewe Director Center for Systems Engineering and Integration CENTACS	

TIME	EVENT	PLACE
	● MANAGEMENT CONTROL TECHNOLOGY	

SESSION CHAIRPERSON: Allan Curry
AIRMICS

Post Deployment Software Support
for Army Defense Systems

Ingrid A. Eldridge
Systems Validation Division
CENTACS

A Decision-Aiding System
for
Software Development Management

Donovan Young
AIRMICS

A Business Approach to Management
and
Control of the Systems
Development Process

L. T. Herrmann
Manager
Systems Analysis and Development
Shell Oil Company

● HARDWARE/FIRMWARE/SOFTWARE TRADEOFFS

SESSION CHAIRPERSON: Dr. Serafino Amoroso
Software Engineering Division
CENTACS

A Tactical AN/GYK-12 Emulator

Edward J. Beach
Systems Validation Division
CENTACS

Technology Upgrade
of
Existing System Peripherals

Jeffery S. Yohay
and
Martin I. Wolfe
Software Engineering Division
CENTACS

TIME	EVENT	PLACE
	A Case Study of the Software/Firmware Development for a Microprocessor-Based Computer	
	James E. Scott Missle System Software Center U.S. Army MIRADCOM Redstone Arsenal	
	● GRAPHICS	
	SESSION CHAIRPERSON: Dr. N. Radhakrishnan COE	
	GRAPHICS STANDARDS	
	Bertram Herzog Director University Computing Center University of Colorado	
	GCS AND GRAPHICS Standardization in the U.S. Army Corps of Engineers	
	James M. Jones II Research and Development Software Group ADP Center	
	Three Dimensional Geometry Software Group with a Practical Application	
	Fred T. Tracy U.S. Army Corps of Engineers	
	Coherence Concepts in Computer Synthesized Real-Time Displays	
	John Staudhammer* U.S. Army Research Office *(On leave from North Carolina State University)	
10:00-10:30 AM	BREAK	

TIME | EVENT | PLACE

10:30-Noon | PLENARY SESSIONS |

- TESTING THROUGH FORMALIZED METHODS OF REQUIREMENTS AND PROCEDURES

SESSION CHAIRPERSON: Norman J. Taupeka
Chief
Systems Engineering Division
CENTACS

A Structure for Developing Verifiable and Validation Software Systems

Jon C. Jervert
Systems Engineering Division
CENTACS

Verification and Validation of Tactical Systems

LTC. Charles R. Lindsey
and
Joseph W. D'Oria
CORADCOM

Software Testing at the System Level

J. Gary Nelson
Headquarters
U.S. Army Test and Evaluation Command

- AUTOMATIC TEST AND DIAGNOSTICS

SESSION CHAIRPERSON: Milton Tenzer
CENTACS

OPAL - A Mondern Language for Test Programming

Helmuth M. Kaunzinger
Software Engineering Division
CENTACS

Automatic Generation of Test Programs in Atlas

Noah S. Prywes
Professor of Computer Science
University of Pennsylvania

Some Issues Related to Computer System Built-in-Test

J. B. Clary
A. R. Jai
Research Triangle Institute

TIME	EVENT	PLACE
	● COMPUTER ARCHITECTURE/STANDARDIZATION/COMMONALITY	
	SESSION CHAIRPERSON: Frank E. Ward CENTACS	
	Life-Cycle Cost Analysis of Instruction-Set Architecture Standardization For Military Computer-Based Systems	
	Harold S. Stone University of Massachusetts and Aaron Coleman U.S. Army CORADCOM	
	A Proposed System of Buses For the Military Computer Family	
	William E. Burr U.S. Army Communications Research and Development Command Fort Monmouth	
	Evaluation of Alternative Computer Architectures An Overview of Three Studies	
	William Dietz Department of Computer Science Carnegie Mellon University	
	● RELIABILITY/SURVIVABILITY	
	SESSION CHAIRPERSON: Dan Hocking AIRMICS	
	MUTATION ANALYSIS: Recent Progress	
	Dr. Richard DeMillo School of Information and Computer Science Georgia Institute of Technology	
	Adaptive Testing	
	R. L. Stone General Research Corporation	
Noon	FORMAL END OF SYMPOSIUM	
Noon-3:00 PM	OPEN FORUM	Spencer's Annex

EXECUTIVE SUMMARY

By

Dr. Stanley M. Taylor
Symposium Chairman

The Second Software Symposium, sponsored by the US Army Computer Systems Command via the Army Integrated Software Research and Development (ISRAD) Working Group, was held on 25-27 October 1978 in Williamsburg, Virginia. The ISRAD Working Group, established in 1974, having representatives from development agencies which are performing computer software research and development, and using agencies, particularly Program Managers' Offices, etc., were invited to participate in working group meetings to provide requirements guidance and to identify problems from the users point of view.

The objectives of the ~~ISRAD Working~~ Group include the following:

- To identify Software Research and Development Goals which satisfy the needs of the using agencies.
- To respond to the software needs of Army Project Managers and Systems Developers by concentrating on software development and maintenance problems.
- To identify unnecessary duplication of effort and gaps in Research Programs which need to be addressed in ongoing and planned R&D efforts.
- To ensure that Software Research and Development is made visible and addressed separately from other R&D projects.
- To provide the Army focal point for coordinating Software Research and Development.
- To serve as a forum for information exchange on Software Research and Development, and to ensure that results of Software Research and Development efforts are disseminated to potential users.

This Symposium series provides one of the many vehicles utilized by the ISRAD Working Group in the accomplishment of each of these objectives.

The Army's Second Software Symposium was a successful continuation of the precedents set in the objectives for the first symposium held in May 1977, namely:

- Provide recurring formal and informal meetings of the members of the ISRAD Community -- managers, developers, and users -- with emphasis on an appropriate environment for informal dialogs which establish mechanisms for continuation of technical exchanges between members of the community.

- Present recent developments in the Software R&D Programs -- including particularly those areas which have received little or no emphasis in current programs.

- Develop an ISRAD Community awareness.

- Provide an access to the ISRAD R&D programs.

The Symposium Organization Committee, under the aegis of the ISRAD Working Group, met approximately eight times over a period of fourteen months to develop the agenda, determine attendees, select speakers and design the format by which to process the content of the two-day meeting. The format of the Symposium was tailored to highlight the DOD Software R&D Technology Plan (September 1977). However, several areas not addressed by the DOD plan, but which were deemed by the ISRAD Working Group to be important enough to be included, were incorporated into the final agenda. The additional program areas are concerned with Security, Human Factors, and Graphics. These latter areas represent particular areas which were largely ignored in the Technology Plan, but it is the recommendation of the ISRAD Working Group that they should be included in further evolutions of that Plan.

The two-day Symposium was preceeded by a special four-hour microprocessing workshop entitled "Microcomputer Tutorial -- The View Looking Down", presented by Professors James Gault and Wesley Snyder from North Carolina State University. This workshop proved to be extremely popular as well as informative to those able to attend it.

The Symposium was organized in a series of plenary sessions and a series of parallel technical sessions in order to cover the broad spectrum of topical areas selected for this year's agenda. BG Riley, Deputy Command, USACSC, made the welcoming address, identifying the above-mentioned objectives and introduced the Symposium Chairman, Dr. Stanley M. Taylor of Ballistic Research Laboratory, ARRADCOM/APG, Maryland. Dr. Taylor, in turn, introduced the two keynote speakers, Mr. Barry DeRoze, TRW, formerly with the DDR&E of DOD, and Mr. Mark Grove, his replacement.

It should be noted that the choice of keynote speakers was in consonance with the general theme of the Symposium. Mr. DeRoze traced the evolution of establishment of the DOD Defense Systems Software Management Program, and the resultant evolution of the requirement for centralized program monitorship within services which led to the development of the Army ISRAD Working Group and its R&D Program, as the Army response to this requirement. As one of the principal authors of the DOD Plan, Mr. DeRoze presented the background behind the plan as well as his perception not only of its impact on specific defense systems and on industry policies, practices and procedures, but also presented an assessment of application of the plan, from both the government and industry viewpoints. Emerging issues of standardization and microprocessor management were identified.

Mr. Grove confirmed the intent of the DOD to continue close monitorship of DOD Software R&D activities and the need for such a program to provide DDR&E with backup data for Congressional justification of R&D programs in this broad area of technology development. Mr. Grove noted that practically every significant modern defense system is reliant upon digital computation elements or subsystems for either its native operation or its integration into the tactical, strategic or C^3I environment. Consequently, a major fraction of the acquisition costs of modern systems is going for the associated embedded computer resources and the similar fraction of the logistic support cost is devoted to this element of any given system. He noted that DOD has established a policy initiative with DODD 5000.29 which mandates the use of higher order languages (HOL) for preparation of defense system software for embedded computer resources. The objective is to get a grasp on the estimated $3 billion annual expenses for software and to improve the availability and modifications/update/turnaround times. DODD 5000.31 takes the policy a step further and restricts the selection of allowable HOL's for future systems to a manageable few from the hundred or so now in use.

Mr. Grove noted that there is almost universal acceptance of the need for improved discipline in the development, acquisition and support of embedded computer resources, but these new policy initiatives have met the not unexpected noticeable but tractable resistance always experienced by changes in policy. The next step in the strategy is via DODD 5000.xx, to exercise similar policy initiative to explore follow-on in the area of computer architecture. Mr. Grove emphasized that the message of these policy directives is a simple one: Optimum decision on a per-system basis do not necessarily lead to either the best or most effective force/system, when the full life cycle is considered. In general, Mr. Grove noted, the principle of the use of prudent management discipline across the total DOD program with respect to the totality of embedded computer resources appears sound. Mr. Grove further noted that it is his expectation that the DOD Software R&D Technology Plan will remain a living document and continue to evolve as experience dictates.

MG Buckingham, Commander, USACSC, made the banquet address, highlighting the future requirements of meshing of communications and data processing and further noted that the areas of technology which underpin the Software R&D requirements of the Army and DOD are in a most difficult transition period, from that of what has been largely initial and necessary discovery type research to one of a more disciplined nature. He challenged the Army R&D community to make a 'quantum jump' in efforts to convert programming from an art to more of an engineering science -- particularly in the applications area of interest to the developers of Army and DOD weapons systems.

SUMMARY OF FINDINGS

Presentations evoking the most interest were the plenary sessions on Security, Human Factors, Graphics and Life Cycle Management.

- Security: The session highlighted the need for Army users to develop detailed security specifications for their systems in order that technological design can be tailored to their requirements. (The DDR&E Technology Areas did not at the time of the Symposium include security in its areas of emphasis. However, the need and interest within the Army exists and, accordingly, this session was a significant part of the agenda).

- Human Factors: The contents of this session focused on factors influencing performance, especially productivity, in software development. Prescriptive aspects of project management such as team organization and use of well-defined software development techniques. It was suggested that, within ISRAD, multidisciplinary working groups be formed to determine human interface requirements early in the system development cycle.

- Graphics: This session presented graphics work in two areas that are pertinent to the Army mission. The first area concerned interactive computer graphics applications in 3-D geometry generation and display. One presentation described a graphics software package that provides an engineer with the capability of generating and editing three dimentional structures. Another, by Dr. Staudhammer, described a display device and supporting software for the display of color and black and white three-dimentional objects directly from a mini-computer to studio-quality television. The second area covered the necessity of developing a set of standards for graphics software.

- Life-Cycle Management: Several aspects of this session reaffirmed the findings and recommendations of the Second Software Life Cycle Management Workshop. Topics put forth and discussed here included a call for construction of standardized definitions, terms, and classifications or taxonomies. A systematized view of the life cycle structure was presented and linked to the use of automated management tools.

SUMMARY OF RECOMMENDATIONS

The following are consensus recommendations expressed by the Symposium leadership, participants and attendees. In line with the general objectives of the ISRAD Working Group, these recommendations should reflect on and have impact on potential future state-of-the-art of the ISRAD Program. In the future, the Army should:

- Expand upon the thesis of MG Buckingham's challenge to accelerate the development of software and programming skills from an art to more of an engineering science.

- Include the concept of networks and their attendant problems in future research programs, as recommended by MG Buckingham.

- Conduct a Human Factors review to cover human/machine user interaction and also language design.

- Analyze the role of the user (frequently the Project Manager) in protecting his interest during rapid technological growth -- that is, how to keep the technological process from subverting the needs of the users, while at the same time providing as much of the new technology for his requirements as is feasible.

- Study ways of cost reduction in the field of software engineering.

- Emphasize meshing of communications technology requirements with the software development.

- Enhance compatibility of software R&D among various Army organizational elements.

- Identify problems in estimation and development of resource requirements and testing validations procedures as applied to management information systems.

- Identify problems in estimation of life cycle costs over the entire life cycle of the proposed system -- include training costs as well as resource requirements to accomplish life cycle maintenance.

- Identify problems with interoperability between various systems which must be integrated in any overall battlefield environment.

Finally, it may be stated that the Symposium was an unqualified success in meeting its objectives. That is, the dialogue between various components of the development community and the user community within the Army did take place. We trust that the technical and programmatic exchange will continue on an informal basis until the next Symposium.

E. Marie Smith of International Business Services welcomes new attendees to the registration area.

Dr. Stanley Taylor, Symposium Chairperson and **Mr. Mert Batchelder** of USACSC discuss upcoming events of the first day.

Symposium participants, **Jean Hooper**, **Norm Taupeka** and **Ray Sidorsky**, examine the agenda.

Jean Hooper delivers her presentation on "A Review of Factors which Influence Software Development Performance."

Attendees in a general session.

MG Clay T. Buckingham, Commanding General of USACSC, was the Banquet Speaker.

MG Buckingham answers questions.

Dr. John Staudhammer, Deputy Symposium Chairperson and Planning Committee member **LTC Roy Busdiecker** chat with **MG Buckingham** after his banquet speech.

FUNCTIONAL SYSTEMS REQUIREMENTS

COL Ray Ketchum
TRADOC

SESSION CHAIRPERSON: COL Ray Ketchum

TRADOC

SESSION SUMMARY

This session highlighted activities underway within the computer development community to describe the architecture and environment within which Battlefield Automated Systems will operate. It also addressed TRADOC (Army) efforts to control proliferation of battlefield systems and described their relationships (interface requirements).

The first paper (Army Battlefield Automation Architecture) was by Major James H. Helberg and Mr. Dennis P. Mahoney. Its relevance was to describe for the Symposium the approach to developing an automatic architecture as a part of the overall battlefield system.

The second paper (Army Battlefield Interface Concept) was by Major M. W. Robinson and Captain John T. Ratzenberger. Its relevance was to describe for the Symposium the study of the interface/interoperability requirements among battlefield automated systems and the relationship of the study of automation architecture and system development.

The third paper (Techniques for Controlling Proliferation of Automation on the Battlefield) was by Major Richard D. James and Dr. Edward R. Fowler. Its relevance was to inform the Symposium of the Army's methodology of controlling the proliferation of automation on the battlefield.

Army Battlefield Automation Architecture

Major James H. Helberg
Mr. Dennis P. Mahoney

USACACDA, Fort Leavenworth

1. An automation architecture is conceptually unconstrained by systems, organizations, or procedures unless automation was considered in their development. Traditional frameworks may be modified considerably by an automated systems network. Therefore, a framework must be defined in which automation can be supportive of corps objectives. Basic to describing such a framework is the definition of information requirements. This will lead to a description of information flow and suggest the rationalization of certain functions within and among functional systems. A basic sequence of questions must be answered. CACDA is taking steps to define the functional information requirements as well as the information flows.

2. To achieve a practical automation architecture, as well as meeting the immediate need for automation to support the field, it is necessary to field certain BAS although they do not fit into an architecture and then bring a comprehensive architecture on the scene later. To begin this process, essential interfaces are being defined by CACDA in the Army Battlefield Interface Concept.

3. Essential to architectural development is the inclusion of certain operational design criteria, the most important of which is standardization. Lack of standardization is a major problem preventing development of an automation architecture. Interoperability, CONOPS, security, RAM, and other design criteria depend on standardization in the areas of hardware, software, data elements, etc. The Army is embarked on several actions to address standardization.

ISRAD
BATTLEFIELD SYSTEMS ARCHITECTURE

Mr. Dennis P. Mahoney
Battlefield Automation Management Directorate
United States Army Combined Arms Combat Development Activity
Fort Leavenworth, Kansas

Within the past several years, there has been a dramatic increase in attempts to apply automatic data processing technology to problems of military command, control, communications, and related functional areas. This increase and the resulting rapid development of ADP systems has caused problems in compatibility and interoperability to become potential roadblocks to effective use of automation on the battlefield.

The purpose of this paper is to describe ongoing efforts to develop an architecture for battlefield automation as a part of an overall systems architecture and to relate it to other parts of such a systems architecture to include information, communications, and management.

The term "systems architecture" is defined by the U.S. Army as follows:

SYSTEM ARCHITECTURE - The generalized description and portrayal of a functional system composed of several interacting/interoperating subsystems arranged in such a manner as to satisfy the requirements stated in an overall concept.

When the Army speaks of systems with respect to an architecture, it commonly refers to information, automation, communications, and management, all working from some standardized foundation. Although there may be other minor considerations with respect to systems, these are considered to be the four major elements in a systems architecture.

In order to allow development of a systems architecture, certain questions must be answered. These might be summarized in the following list:

1. What jobs must be done?

2. Where are the jobs done?

3. What information is needed to do the job?

4. What is information flow pattern on the battlefield?

5. What should be the architecture for integration of functional systems.

6. Where and how should automation be applied?

It has been the experience of the Army too often to have dealt primarily with question six (6) without completely having answered questions one (1) thru five (5). From these questions, it may be perceived that efforts to apply and integrate automation on the battlefield hinge on two areas not specifically dealing with automatic data processing (ADP)---those being function and information.

To set the stage for this discussion, a convenient reference framework will first be established. In order to achieve tactical units objectives, the commander of each unit and his staff must perform certain basic functions. He must see the battlefield, plan the operation, allocate resources, fight the battle, and sustain the force. These functions can be arrayed in a matrix with the battlefield functional systems currently used by the U.S. Army Training and Doctrine Command (TRADOC) in the Battlefield Automation Management Program (BAMP). These functional systems are maneuver, field artillery, air defense artillery, air/ground, engineer, electronic warfare, intelligence, communications, command and control, logistics, and administration. Each center or school within TRADOC has been charged with developing functional system concepts for each of these functional systems.

With the five commander and staff functions arrayed on the vertical axis and the eleven battlefield functional systems listed horizontally, the way automated systems are generally applied to the battlefield at the present time can be described. This approach, no matter how functions and functional areas are described, basically automates current manual procedures but does not provide for the exchange of information between the functional systems, nor does it provide for integration of functions across the battlefield. The result is a large number of stand-alone ADP systems, many of which support similar functions in different functional systems.

In order to make such an approach work, extensive interoperability between the automated systems involved is required. Unfortunately, virtually every system has its own data elements, language, applications, hardware, and so on. Interoperability is, therefore, almost impossible without significant expenditures for translational devices or so called "black boxes."

A better approach might be to integrate like functions across functional areas/systems. This means that similar functions would be rationalized and users in any of several functional systems would have access to the same ADP applications in support of their functions. There are several advantages to this approach: It minimizes the number of computers on the battlefield; it rationalizes the development of functional applications; it simplifies interoperability; and it minimizes the burden on funds, people, and communications. Practical problems in communications, technology, doctrine, and organization, however, make this integrated functional approach, at best, a long-term goal.

Thus, two broad problems must be addressed: What can be done now, and what can be done over the long-term to achieve an integrated automation architecture? The first thing that must be done is to recognize current limitations and live with them. While it is widely recognized that a complete systems architecture is needed as soon as possible, it must also be recognized that such an architecture is not available now. The first increment of automated systems fielded will not be an integrated, interoperable, multi-functional architecture. It will be a series of nonstandardized systems which, while perhaps not optimum, provides assistance with some of the pressing problems which exist right now.

The second thing required now is to take positive action towards achievement of a battlefield automation architecture. Recalling the questions raised earlier, these must be answered in about that order to make possible an orderly progression towards such an architecture.

In order to begin, it is necessary to refer back to the matrix described earlier. By adding a third dimension, that of echelon of command, a figure to operate as a tool for identifying information needs on the battlefield can be built (figure 1).

FUNCTIONAL INFORMATION REQUIREMENTS

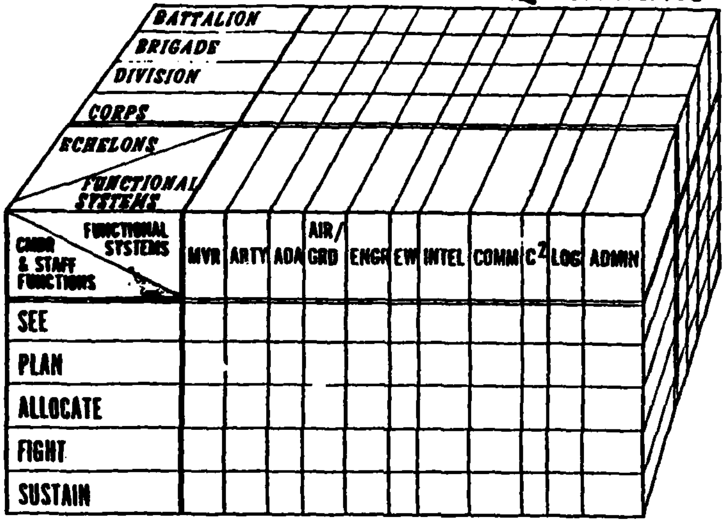


Figure 1

These needs are described in terms of input/output information for every functional area at all echelons, corps and below. This will then serve as a baseline for determining the various paths over which information can travel from one point to another in any combination of commander and staff function, functional area, and echelon and ultimately lead to an architecture for applying automation to the information flow on the battlefield. Thus, the first four questions from the list are addressed, and the Army can proceed with the application of automation to enhance its ability to process and distribute information within a battlefield automation architecture.

Battlefield automation architecture as a subset of a total systems architecture is defined as "....a series of integrated battlefield automated system networks characterized by interface/interoperability, continuity of operations, security, RAM, and multi-functional processing all of which are largely achieved by standardization."

It is important to realize that such networks may cut across traditional frameworks of organizaiton, branch, and echelon. In a purely conceptual framework, an automation architecture is unconstrained by systems, organizations, or procedures which are not designed expressly for the automated environment. This requires that a close look be taken at all these factors when designing the automation architecture. The purpose of all Army architectural effort is to determine how the Army can move from where it is now towards the ideal, or at least an environment where automation is rationally applied to enchancment of battlefield effectiveness and does so at the least cost.

As was expressed in the definition of automation architecture, certain characteristics are essential to a network of integrated battlefield systems. The Army calls these operational design criteria (ODC).

The operational design criteria for ADP systems in the Army are interface/interoperability; continuity of operations (CONOPS); security; reliability, availability, and maintainability; and standardization. Each of these criterion is, by itself, a major area of concern to users, developers, and commanders at all levels and must be addressed at every stage of the development cycle. Additionally, they are closely related so that requirements for each must be evaluated for the effects on the others.

While each criterion is important, one may hold the key to the Army's ability to address the others. That one is standardization. From the Army's point of view, lack of standardization is the major problem area associated with automation architecture development in the U.S. Army. It prevents any real progress being made in interoperability without the use of elaborate translational measures, as well as complicating, beyond probable solution, the measures to assure continuity of operations under a variety of conditions. Security of systems and data bases is also complicated by the variety of standards now used, although the netting of automated systems in a comprehensive architecture presents its own set of security problems.

The most obvious major problem area associated with a lack of standardization is the reliability, availability, and maintainability (RAM) of hardware for a large number of unique systems under battlefield conditions. One survey, conducted by the U.S. Army Communications Research and Development Command, provides some insight into the magnitude of the problem. This survey predicted that by the late 1980's the total population of battlefield computers in only the Army may exceed 135,000 computers of all types. Numbers of this magnitude makes it clear that the RAM problem will be practically insurmountable without standardization.

More subtle, but no less perplexing, are the problems of software support and maintenance. Software support for the U.S. TACFIRE system alone calls for about 100 personnel and $7M annually. Similar burdens are implied for every automated system fielded. Obviously, there is a serious resources problem. Doctrine and organization for post deployment software support must be examined. Standardization is the only way to make adequate support achievable.

What is being done to address automation standardization?

To address the problem of data element standards, the U.S. Navy has developed a data base package for management of data element dictionaries called the record association system/standard data element system for use by all battlefield automated systems proponents and developers. If successful, it will provide a means by which data element standardization can be managed. This is a key consideration in the ability to exchange information between automated systems.

In the area of hardware standardization, the Army is working with the other services towards the development of a military computer family and its associated software to provide for compatibility of procedures, as well as commonality of hardware, for all battlefield automated systems. The magnitude and importance of this project was illustrated earlier when we discussed the numbers of computers projected onto the 1980's battlefield. The key to the approach is one that defines common physical characteristics, connector compatibility, and standard input-output functions rather than one of single sources for equipment.

Progress is being made in the areas of message format and language. Format standards must be continuously and rigorously applied to insure that the message can be understood even when all the other interface elements are present. The Department of Defense is directing an interservice program to address this problem called the "Joint Interoperability of Tactical Command and Control Systems" (JINTACCS).

Development of a common high-level language called DOD-1 is also underway. There are numerous reasons why a high order language must be developed and agreed upon.

Obviously, common computer language makes possible common operating systems and applications programs, thus permitting continuity of operations and rationalization of applications.

More critical, though, is the problem of computer programmers. A study by Texas Instruments, Inc. has predicted that, given current programming methodology, there could be a shortage of as many as 10 million computer programmers in the U.S. by 1985. The use of high order language is a major step to alleviate this shortage by increasing programmer productivity.

Two standardization related areas with which the Army must better come to grips are those of integrated logistics support and source data automation. We need to reexamine our doctrine for integrated logistics support (ILS) in the area of training, supply, and maintenance in order to support battlefield automated systems. In an environment where cross-attachment of units will probably be necessary, we must be able to provide logistical support in automation as in other areas. Training, supply, and maintenance standardization must be addressed to provide the ability to support attached units effectively.

In addition, we must develop compatible systems and procedures for what we call source data automation, that is, those means by which information is originally inserted into the automated system network. These systems and procedures must be standardized throughout the battlefield in order that we might be capable to input data to a variety of different systems when required.

In summary, it remains for the Army to commit itself to solving these problems now (it will cost much more later) in order to make the gains in the effective application of combat power which can result from the fielding of a series of integrated networks of battlefield automated systems---a battlefield automation architecture.

ISRAD
BATTLEFIELD AUTOMATION MANAGEMENT PROGRAM

Mr. Dennis P. Mahoney
Battlefield Automation Management Directorate
United States Army Combined Arms Combat Development Activity
Fort Leavenworth, Kansas

The purpose of this paper is to describe the Battlefield Automation Management Program (BAMP). The Army fully intends to capitalize on automation. It provides us a means to change the odds in a potential war in which we will probably have to fight outnumbered and outgunned by the enemy. However, initial efforts towards automation of the battlefield have encountered difficulties.

In April 1977, at Fort Hood, Texas, Tactical Automation Appraisal II took place. Two major issues emerged. First, there was no single manager in charge of battlefield automation and second, there existed no guiding concept for battlefield automation management.

As a result, the Vice Chief of Staff of the Army (VCSA) directed the US Army Training and Doctrine Command (TRADOC) to establish a single focal point to manage battlefield automation within the following guidance, that of developing a philosophy and a methodology to manage automated systems development, which will control proliferation of computers on the battlefield. From this guidance emerged the BAMP philosophy which is to optimize fighting capability.

With this background, the Combined Arms Center proceeded with development of our methodology model. This model uses three basic inputs to an evaluation process which results in a specific recommendation regarding the development of each battlefield automated system.

The management of battlefield automated systems is complicated somewhat because the Army currently acquires its automation capability under two different regulations, Army Regulation (AR) 18-1 and AR 1000-1. Thus, before implementing the methodology, it was necessary to merge the life cycle phases of each. The result is the set of categories identified as I, II, or III which are Concept/Definition, Validation/Development and Production/Installation. This enables us to track a system from concept to fielding if systems are procured under either regulation.

In implementing our methodology, we require the proponent of a system to provide a series of inputs. The first, the Battlefield Functional System Concepts, are documents prepared by each of the eleven battlefield functional proponents identified in the systems architecture presentation. They describe, as an example, how the field artillery, or maneuver, functions are performed on the battlefield. Their orientation is conceptual and functional, as opposed to being driven by echelonment or equipment. This document provides BAMP analysts insight into the basic actions performed by the functional system, and explains how the system will assist the commander with his requirements to see the battlefield, plan his operations, allocate his resources, fight the battle and sustain his forces. Our second stream of inputs are the information shortfall statements, also submitted by the battlefield functional proponents.

An information shortfall is that portion of the total known information requirement which is currently not being fulfilled. The proponents described each of their shortfalls as either a void, a condition where the information is not available, or as a deficiency.

If the shortfall is a deficiency, they described it in terms of a problem in either timeliness, accuracy or resolution. Resolution, in this case, means that information may be available in a timely manner, with a high degree of accuracy, but still does not portray the desired picture.

The shortfalls evolve from the functional concepts mentioned above as expressions of information required to perform missions or tasks on the battlefield. The first two inputs just discussed provided the jumping off point in our evaluation of automated systems. The evaluation actually began with the submission of our third input -- the battlefield automated system description.

This document, also prepared by the functional proponent, describes in detail a fielded or proposed ADP system. Examples of four battlefield automated systems are the Corps Tactical Operations System (CTOS), Army Terrain Information System (ARTINS), Standoff Target Acquisition System (SOTAS) and Mobile Army Ground Imagery Interpretation Center (MAGIIC). The system description tells how the battlefield automated system is functionally integrated, what shortfalls it addresses, and it describes the systems resource requirements in terms of dollars, people and communications.

From these three inputs, we have a foundation for evaluating battlefield automated systems. The concept describes the arena in which a system will operate -- the shortfall describes the need -- and system description ties it all together. Thus, the integration thread can be carried from concept to actual fielding of any proposed automated system. Using these inputs we filter each proposed battlefield automated system through the system evaluation process.

Initially, we insure the proposed battlefield automated system meets minimum operational design criteria. The system is carefully checked for standardization, which is critical due to the number of systems under development. Specifically, standardization is the key to success in the next two areas shown, which are interface/interoperability and CONOPS, or continuity of operations in the event a system is shut down for any number of reasons. The reliability, availability and maintainability of the system is analyzed as is the systems proposed degree of security.

If the system meets those ODC requirements, it is further evaluated to determine its performance, resource burden and payoff. Proposed battlefield automation systems capabilities are evaluated to determine if they satisfy information shortfalls. Next, the percent of shortfall satisfaction is reviewed and, finally, the impact on operational effectiveness for the proposed battlefield automated system is analyzed.

Next, the system is evaluated with respect to burden on US Army resources and are quantified in terms of dollars, people and communications. Specifically, its total cost, in dollars, throughout the acquisition life cycle is estimated. All costs, to include Research Development Test and Evaluation (RDT&E), Other Procurement, Army (OPA), Operations and Maintenance, Army (OMA), which includes civilian costs, and Military Personnel, Army (MPA) dollars, are compiled.

The burden in people is analyzed to determine impact on operations, maintenance and training. Both direct costs, such as systems operators and post deployment software support personnel, and indirect costs, such as vehicle operators and maintainers, are included in this assessment.

The systems communications requirements are screened to assess their impact. Because of the burden that battlefield automated systems place on US Army communications resources, it is essential to determine whether new technology is required, additions are needed in the form of equipment capability, additional load is placed on current communication systems, or if there is no discernable impact on our current communications systems.

Lastly, the payoff evaluation is determined by assessing the overall net benefit to the Army in terms of information shortfall satisfaction versus the dollars, people and communications costs. In addition, key factors will be highlighted with respect to potential acquisition and operating problems, such as personnel operator qualifications and factors relating to the previously mentioned operational design criteria.

Upon completion of this process, which occurs three times at various points in each system's acquisition life cycle from concept through fielding, the final recommendations are made. The alternatives are accelerate, continue, terminate, modify or rejustify.

To provide some appreciation of the magnitude of the BAMP, there are approximately 70 fielded or proposed hardware and software battlefield automated systems. Each of these systems is to be evaluated through the Battlefield Automation Management Program.

Army Battlefield Interface Concept

Major M. W. Robinson
Captain John T. Ratzenberger

USACACDA, Fort Leavenworth

1. The Army Battlefield Interface Concept (ABIC) is an ongoing procedure and iterative document for identifying, classifying, describing, and consolidating all interfaces among Battlefield Automated Systems (BAS). While ABIC1978 addresses only Army BAS at corps level and below, future cycles will extend the A"IC to include joint and allied systems at all echelons.

2. Under the ABIC, BAS are studied to determine functional information needs and outputs and the methods of information transmittal. This review also encompasses the methodology and requirements of the Battlefield Functional System Concept, and Information Shortfalls and the BAS description. Each BAS is compared to all other BAS and all matching inputs and outputs are documented as potential interfaces.

3. The objectives of the ABIC are to:

   a. Provide an objective automation architecture.

   b. Promote interoperability and standardization.

   c. Rationalize redundant sources and requirements.

   d. Provide general and detailed guidance to system and material developers.

ARMY BATTLEFIELD INTERFACE CONCEPT (ABIC)

John T. Ratzenberger, CPT, U.S. Army
Battlefield Automation Management Directorate
U.S. Army Combined Arms Center
Ft. Leavenworth, Kansas

The purpose of this paper is to tie together the previous two papers - Systems Architecture and the Battlefield Automated Management Program - to show how the Army is moving toward an automation architecture.

An automation architecture is a series of integrated Battlefield Automated Systems characterized by interface/interoperability, continuity of operations, security, RAM and multi-functional processing; all of which are largely achieved by standardization. From this definition, we should lock on two key concepts- Interface and Interoperability - as they are central topics of this paper.

An interface is a boundary or point common to two or more Battlefield Automated Systems or other activities where exchange of data takes place. There are three types of interface:

- Manual extraction, transmission and entry of data from one system to another.
- A remote I/O device connected online to one system at the processing site of another.
- And, an automated exchange of information directly between the central processors of two or more systems, with or without operator assistance.

Interoperability is the capability of Battlefield Automated Systems to directly exchange data in a prescribed format, and to process the data exchanged. By definition, interoperability is achieved only by an automated interface.

Automated systems must interoperate to the fullest possible extent due to the sheer volume - both quantity and frequency - of data to be exchanged on the battlefield. The fragmented system development, the lack of interoperability, and the effects thereof, have been described in the previcus papers. Although the problem had been addressed for many years by a series of meetings and studies, no viable solutions were found - chiefly due to doctrine changes, static procedures, and cumbersome documentation requirements. The clear need to simplify procedures was addressed at a 30 August 1977 meeting at the Combined Arms Center, when both users and developers jointly proposed a dynamic procedure to identify interoperability requirements.

As a result of this meeting, HQ Department of the Army issued guidance for the Army Battlefield Interface Concept, or ABIC, in November 1977. This was to be an iterative document, updated yearly as new systems emerged and requirements changed. Training and

Doctrine Command, as the combat developer, was given responsibility to maintain and upgrade the ABIC based on Army-wide input. Army Materiel Development and Readiness Command, as the principle materiel developer, was given responsibility for the engineering of approved interfaces to meet user requirements.

The purpose of the ABIC is five-fold:

- Create a simple mechanism for HQDA to approve and fund interface and interoperability needs.
- Assist the materiel developer in funding of hardware, software and communications to support specific links.
- Provide a document which identifies requirements for interoperability of automated systems.
- Provide guidance to the proponents to promote interoperability and standardization.
- Last, but most important - provide a comprehensive and objective automation architecture supporting the whole corps battlefield.

The scope of the ABIC is broad, encompassing all Army systems interfaces at corps and below - provided they have an approved Letter of Agreement between the combat and materiel developer - and will be fielded by 1985. Further, all interfaces with Army systems in echelons above corps and with joint, allied and NATO systems will be included. Growth is provided by incrementing the fielding date by one year with each iteration.

The analysis of each system relies heavily on data provided by the proponent. This data describes the system in general, the interfaces with other systems, by type, echelon and area, the interfaces by functional application, and the information exchanged. Each of these will be described using notional examples to illustrate the complexity of the problem.

First, the system description gives a statement of the mission and fielding date. This places the system in perspective on the battlefield and in time. Then the system is described in terms of, essentially, the physical characteristics of the interfaces.

Systems interfaces, as shown in the Figure 1, tell who will exchange data and how. For example, "A" and "E" exchange data via a remote terminal located at "E"; "C" and "D" exchange data via an automated link. "B" and "D" might exchange data - not directly, but through another system. "A" could be an executive system - receiving, storing, and distributing data from a number of smaller, subordinate systems. Note that subordinates can and do exchange data directly. More than one executive could be in the network - this is illustrated by making both "A" and "D" executive systems. A prime example of this might be to make "A" a division command system and "D" a division artillery system.

Interfaces classified by echelon of employment show who owns and operates the system, as illustrated in Figure 2. Comparing figures 1 and 2, note how the five systems have grown to seven as two of the systems - "A" and "C" - are employed at different echelons. Note also the modification of interface requirements as illustrated by systems "B", "C" and "D". Systems "C" and "D" interface directly only if they are both employed at division level - and "B" and "C" interface directly only between battalion and brigade levels. It can also be seen that the division level system "A" must interface with its counterpart at corps.

Interfaces classified by geographical area of employment show the communications requirements to support the interface, as shown in Figure 3. The prime example to note here are systems "A" and "E" which are corps systems by echelon, as shown in Figure 2, but system "E" is employed well forward in a battalion area.

Interfaces are next described by their functional application. Functional applications show the type of data exchanged in terms of basic military information needs such as enemy situation, friendly situation and supplies on hand. Figure 4 (above the dash line) shows two systems exchanging information about enemy forces. For example, command system "A" sends data on enemy forces to up-date the target list in artillery system "B". After the target has been attacked, system "B" sends the results back to "A" to update the enemy situation. Many systems are multi-functional as shown by the entire Figure 4. In this case, artillery system "B" also provides command system "A" with ammo useage data to update both the friendly situation and the ammo supply picture.

The last thing looked at is what information is exchanged in relation to the functional application, as shown on in Figure 5. In this case, the functional application exchange is broken down into specific data elements. The data sent (denoted by "S") or received (denoted by "R"), depends upon what the system has available or requires. For example, command system "A" designates a target to artillery system "B" by providing data elements describing the subject (such as "fire mission"), the unit size (such as "company of infantry"), the activity of that unit (such as "moving in the open"), the date and time they were seen, and the location. "B" would return the results of the engagement by reporting appropriate data elements, to include updating any changed data for the enemy situation.

Now that we have seen what information we have to work with, the analysis process will be described. Each system is looked at in terms of the approved systems requirements and information shortfalls, as discussed in the BAMP paper, to see if the system fulfills corps automation objectives. During this analysis, certain questions are posed:

- What are the minimum data exchanges necessary for effective system operation?
- Can system efficiency be augmented by data exchanges?
- What data must be exchanged to satisfy the commanders information requirements?
- Can the system provide the data?
- Can the system absorb the data?

In the end, all Battlefield Automated Systems are compared to each other and all matching inputs and outputs are documented. This becomes the ABIC product, a document showing - in system by system pairs - a graphic portrayal of all interfaces, a presentation of information exchanged, and supporting rationale for each interface and exchange. The ABIC is used as a requirements document for HQDA approval of interfaces, as supporting info for the materiel developer - and most importantly, to define the corps automation architecture.

During the first iteration of the ABIC, many insights into the paths and pitfalls facing an Army automation architecture were gleaned. These fall into the broad categories of fielding dates, executive systems, software, data base management, standardization, and continuity of operations.

Fielding dates impact in two ways. First, when many systems are fielded at the same time, and second, when systems in a network are fielded at different times. When many systems hit the field at the same time, there is a massive increase in data exchanges. Communications and processors in a network must be able to handle this increase - without an unacceptable degradation of performance. To minimize the problem, systems must be well designed and data base management techniques must be up to speed before-hand. Major changes to software after the fact are too expensive to be a viable solution. Differing fielding dates cause other problems as there may be significant gaps in a network. Cost and operational need must be compared to decide if interim links must be created of if certain links can be delayed. This must be done before fielding as it is cheaper to design interim links into a network than make fixes afterwards. Another aspect of this is continuity of operations - alternate plans must be made now to back-up a network or system if the designated back-up system is not yet fielded.

Executive systems are the best example of the massive increase in information exchanges within a network upon fielding. Again, data base management techniques must be well in hand to ensure the executive systems can handle their mission. Due to their size and complexity, the information to be used must be pinned down early to avoid expensive fixes. On the other hand, due to the key role of the executive systems, it is best to concentrate upon them as it will be much cheaper to change a few executives than a lot of subordinates.

Software must be well-designed and tested from the start - the cost of changes is expensive in both time and money. It must also be designed for ease of maintenance after fielding. Software is really no different than a tank - if it breaks down, it must be quickly fixed to restore operational readiness. The ability to do this, particularily in terms of people and money - is already one of the biggest questions in the automation architecture scheme.

Standardization of hardware, software, data elements and procedures - just to name a few aspects - is the key to an automation architecture although not the total solution. However, standardization for its own sake is not the answer - the future must be planned for to avoid the prohibitive cost of changing horses in mid-stream.

Data base management techniques are of prime importance due to the volume of data involved. The dual capability of automatic and on-line file maintenance must be built-in to prevent the volume of information from degrading the system. A viable network management and control scheme and the decision to use centralized or decentralized data bases must be balanced by the CONOPS requirement to reconstruct data bases quickly.

Continuity of operations is the most critical aspect of an automation architecture. As the Army becomes more automated, the less it can revert to manual operations for back-up. Types and methods of back-up must be identified and integrated into planning and development from the start. In a combat operation, the ability to restore a command and control system to operation quickly may spell the difference between victory and defeat. Additionally, it must also be known if a system can back-up another without degrading the performance of either or both - and whether or not subordinate systems can function without the executive.

As CONOPS is the ultimate tie-in of all other topics, one thing must be made clear. These wartime systems are being designed and tested in peacetime. It may not be possible to completely simulate the intensive demands combat will place on a system or network - but there may not be time to do a better job later.

It is unfortunate that development times, defense considerations and the need to employ certain systems as soon as possible hinder a logical fielding plan. The ABIC is an attempt to pin down interface requirements and provide a comprehensive automation architecture for coordinated development. Upon its success rides the future of Army automation.

ARIC

SYSTEMS INTERFACES

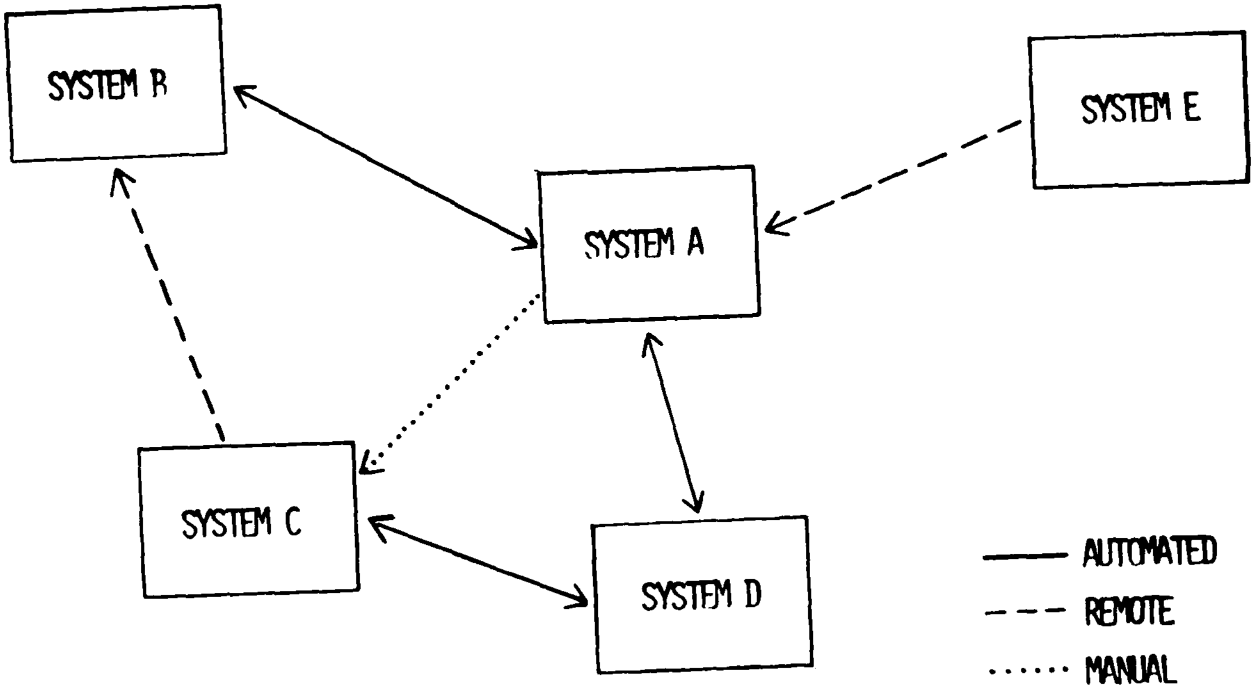


Figure 1.

A P I C

INTERFACE BY ECHELON

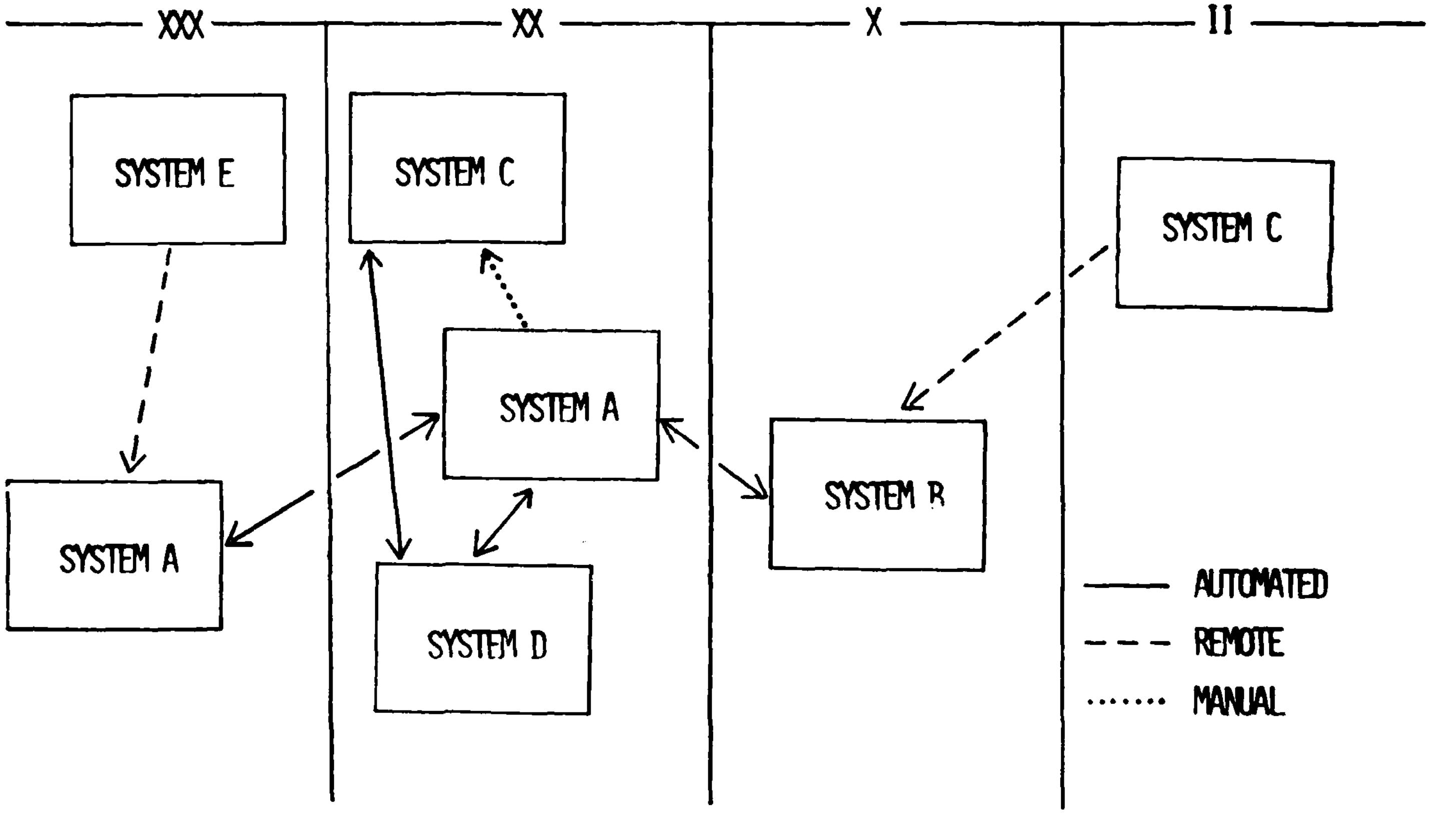


Figure 2.

A R I C

INTERFACE BY GEOGRAPHICAL AREA

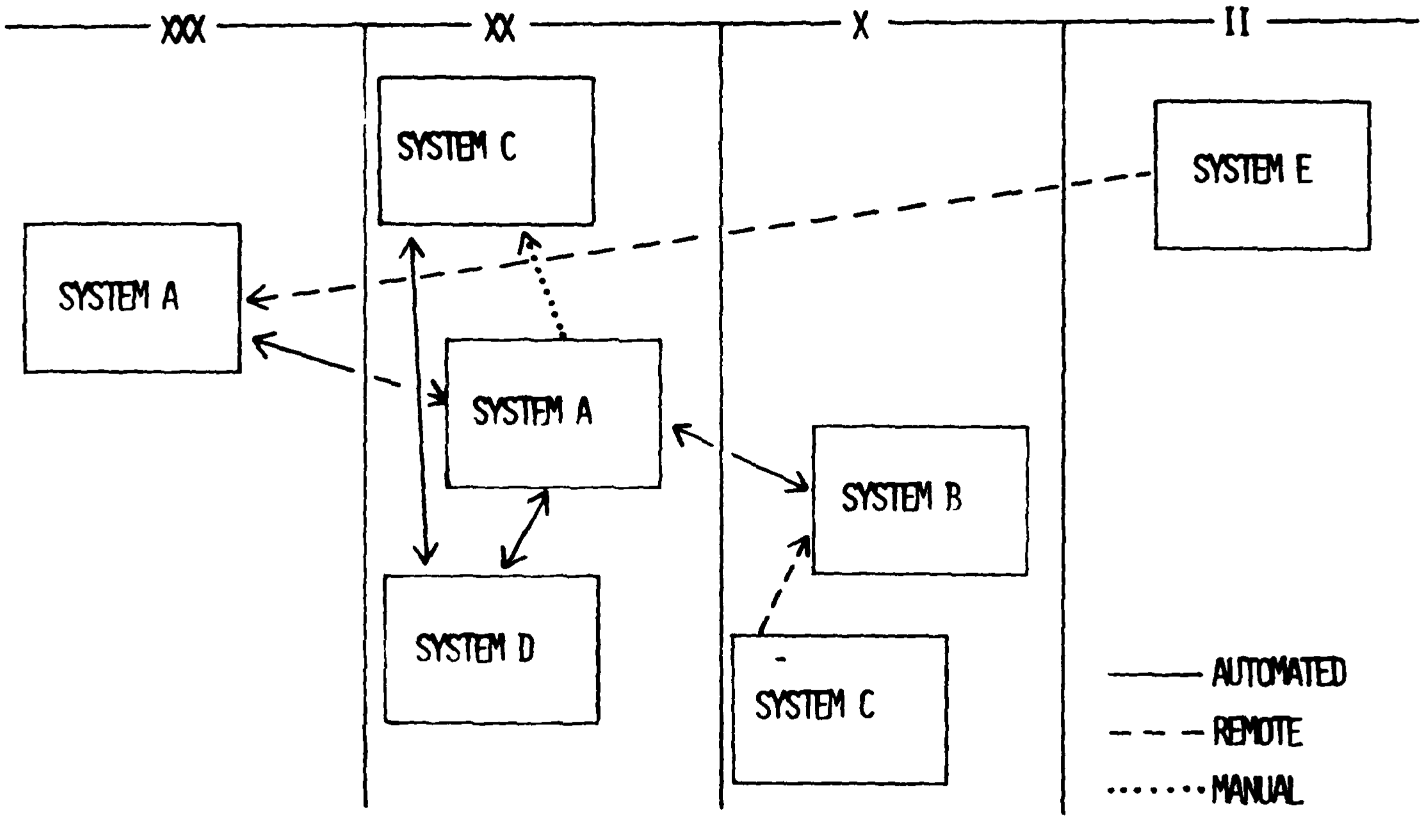


Figure 3.

A R I C

APPLICATIONS INTERFACES

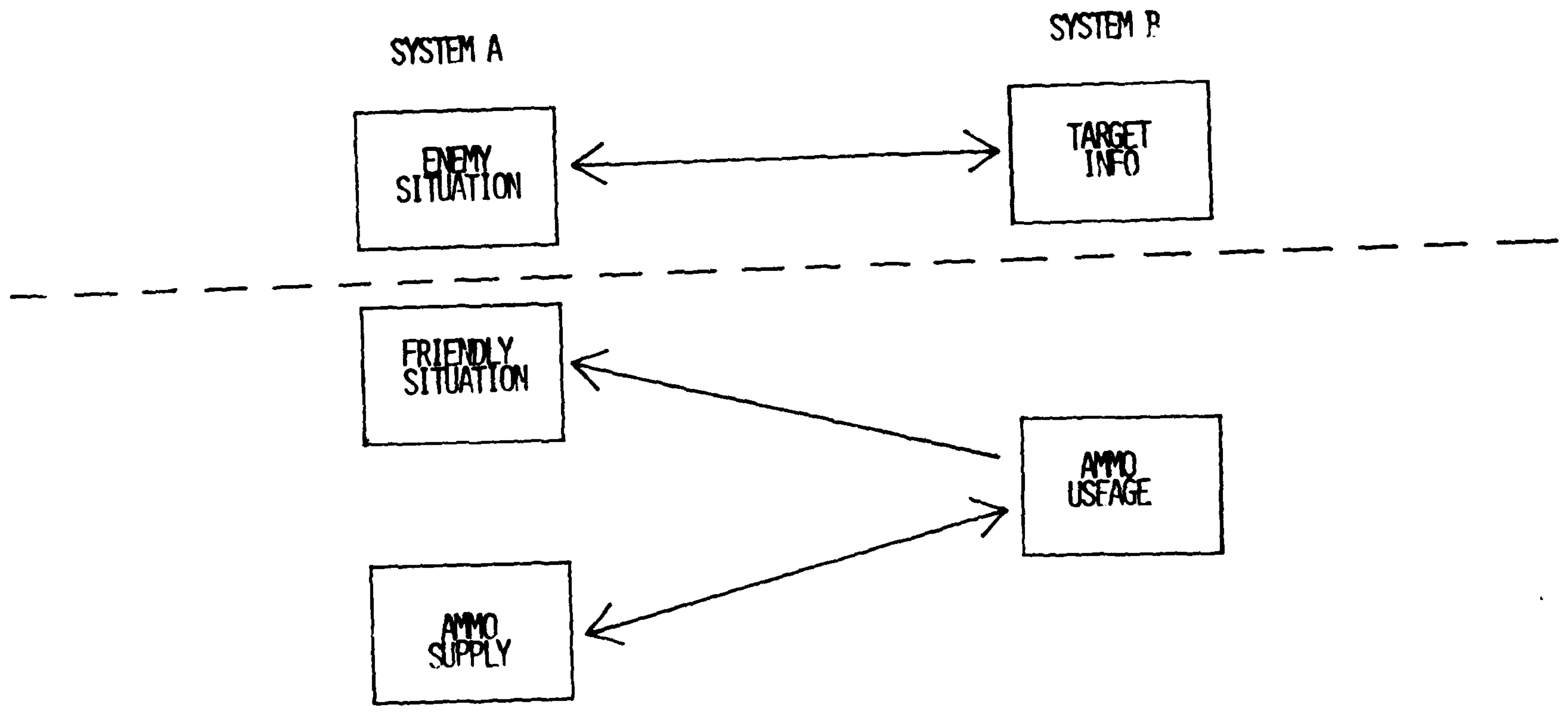


Figure 4.

ARIC

INFORMATION EXCHANGE

SYSTEM A APPLICATION X	INFORMATION ELEMENT	SYSTEM B APPLICATION M
S	SUBJECT	R
SR	SIZE	SR
SR	ACTIVITY	SR
SR	DTG	SR
R	ENGAGEMENT	S
SR	LOCATION	SR

S = SEND R = RECEIVE

Figure 5.

HUMAN FACTORS

Jean N. Hooper
ARI

HUMAN FACTORS

SESSION CHAIRPERSON: Jean N. Hooper

Army Research Institute

SESSION SUMMARY

The Human Factors Session at the Army Software Symposium was focused on two areas where human performance is critical to system operation--the roles of the human as system developer and as system user.

General issues in the design of computer systems to facilitate human-machine interaction were addressed. Clearly, there is a need for cooperation between system developers and human factors specialists to develop interface guidelines that transcend specific systems.

Use of interactive query language systems by naive users to store, manipulate, and retrieve information is becoming more widespread in the Army. Human factors issues in the design and use of query languages were discussed.

Software development is widely recognized as a costly, often error-prone activity which contributes to system unreliability. Recent research on the performance of the individual software developer have been reviewed with the goal of improving the efficiency and accuracy of the programming process.

Human Performance in Software Development

Jean N. Hooper

Army Research Institute

Software development is well recognized as a costly, labor-intensive activity. Improvements in the software development process must focus on the performance of the individual programmer in writing code; unfortunately, it is only recently that the individual's performance has been examined. Different metrics of performance (e.g., lines of code, errors, product quality) were discussed, and research on programmer performance will be summarized. Concluding remarks addressed possible means of improving the performance of the individual software developer.

A REVIEW OF FACTORS WHICH INFLUENCE SOFTWARE DEVELOPMENT PERFORMANCE

Jean N. Hooper
US Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Software production is widely recognized as a costly, error-prone, labor-intensive activity. Many factors which influence the performance of the professional software developer have been identified through controlled research and project audits. Literature in this area will be reviewed to identify factors affecting software development performance, especially productivity. This analysis will be used to generate prescriptive suggestions useful to project management personnel in optimizing performance.

Some of the factors which have been found to influence the performance of the software developer are shown in Table 1. These factors can be grouped into five categories:

a. Project factors, such as program type and complexity;

b. Environment factors, such as turaround time;

c. Programming tools and methods;

d. Management factors, such as programmer team organization; and

e. Personnel factors, such as experience.

This list is not intended to be exhaustive; clearly, just in the area of personnel factors, there are many more such as motivation and skill level.

Project Factors

The effects of project factors on performance are summarized in Table 2. Both Brooks (74) and Zelkowitz (78) cite similar figures of productivity, measured in lines of code (LOC) per year, for software projects of different types. The most difficult type of software is a control program, with productivity rates of about 600 LOC per year. For systems programs, the productivity rate goes up to around 2,000 lines of code per year, an increase by a factor of three. Applications programs are generally written at a rate of 6,000 lines of code per year, roughly a ten-fold increase in productivity over control programs.

Brooks (74) has also offered information on the effect of project complexity on project cost. Note that he uses cost, and not some other metric of performance as his measure, but in the context of his report cost can be roughly equated with productivity. He has determined four different levels of complexity, with level 1 the simplest and level 4 the most complex. A level 1 program, at the simplest level, is produced for the author's own use; Brooks assigns this a cost factor of one. The next level of complexity is a programming system, which is a generalized and documented program, and is three times more costly to produce than an "own use" program. A programming product, at the third level of complexity, is one which requires integration of components, costing six times more than a level 1 program. At the highest level of complexity is the programming systems product which requires everything--it must be generalized, documented and integrated, and costs nine times as much to produce as a level 1 program. Thus, according to Brooks (74), complexity of the program can produce a nine-fold increase in cost.

Project size is another factor which influences productivity. Johnson (77) audited sixteen software projects and concluded that over three times more debugged, implemented, and documented source code was produced per year on small projects. Unfortunately, he neglected to define "small" and "large" projects, thus providing little quantitative information on project size.

In addition to the factors of program type, complexity and size, the required interaction among system components also influences productivity. Brooks (74) notes that projects with high interaction requirements have lower productivity rates.

Environment Factors

The computing environment is another area which may influence the performance of professional programmers (see Table 3). Sackman et al. (68) studied the performance of professional programmers using time-sharing and "simulated batch" systems. The simulated batch system had a fixed turnaround time of two hours. No significant differences were found between the timesharing and simulated batch conditions on any of the performance measures. This can be attributed to the large individual differences in performance which were observed; the individual differences exceeded differences due to the programming environment. The authors then transformed the scores to reduce the variance and used programmer coding skill as a covariate; in this analysis, a significant difference was found in debugging time, with lower debugging time for programmers using the timesharing system.

Turnaround time, and not the use of batch vs. timesharing, may be the factor which influences performance. This is supported by the failure of Sackman et al. (68) to find significant differences on most of the performance metrics employed despite statistical reduction of individual differences. In the Sackman et al. research, the turnaround time for the "simulated batch" system was very short, and was not variable. Rarely are such conditions found in a batch system. Oliver (78) has noted that turnaround time is a factor influencing performance; installations with short turnaround times have higher productivity rates.

Management Factors

Management factors which influence performance are summarized in Table 4. The organization of programmer groups is a factor studied extensively by Scott (73) and Scott and Simmons (75). These researchers developed a communications model of team organization based on the analogy of a programming team to a multiprocessor communications network, with a single instruction multiple data stream (SIMD) organization. Inputs to the model consist of an activity profile and productivity level for each member of the programming group. The activity profile is the percentage of time the programmer spends in productive, personal, and communications activities; these figures are based on empirical data (see Scott, 1973 or Scott and Simmons, 1975, for additional information). A scaled productivity value was the output of the simulation using this model.

Scott (73) and Scott and Simmons (75) simulated the productivity of three different programming group structures, shown in Figure 1. From left to right, these will be referred to as the subteam, traditional, and egoless structures (Weinberg, 1971). With identifical activity profile and productivity inputs for each of the three structures, the egoless and substeam structure groups were significantly more productive than the traditional team structure. No significant differences in productivity were found between the subteam and egoless structures. Scott (73) attributed the lower team productivity of the traditional organization to an information bottleneck caused by the position of the manager of the team. In the other two organization structures, the information flow was distributed across team members.

Using the same model, Scott (73) investigated the effect of team size on the productivity of the traditional group structure. Starting with a three member group, performance of the team was simulated, and team members were added in increments of three up to a total group size of eighteen. The simulation confirmed that there is a point beyond which the addition of personnel provides no increase in group productivity. The productivity increased with the addition of personnel up to a group size of twelve; beyond that point, there were no significant gains in productivity with increased group size. This findings may be related to Brooks' Law (74): adding more personnel to a late software project makes it later, due to the communication and training requirements placed on the personnel already assigned to the project. In Scott's simulation, it is likely that the information bottleneck observed in the traditional team organization reached a critical level with a group size larger than twelve. Unfortunately, Scott did not investigate the effect of group size on the other two organization structures.

In a third simulation study employing the communications model and the traditional team structure, Scott (73) investigated the effect of a highly productive individual on group productivity. Productivity of the group was higher when the productive individual was a group member than when such a person was placed in the role of group chief. Once again, this is attributed to the communication requirements imposed on the team chief which reduce productive time, especially in the case of the traditional team structure used in this simulation.

Weinberg and Schulman (74) performed two experiments to investigate the effect of explicit project goals on performance. In the first experiment, two groups of professional programmers were assigned identical programming problems. One group was given the goal of maximizing efficiency of the program, while the other group was given the goal of minimizing development time to completion of the program. Performance on the task was measured as the number of runs to completion of the program and execution efficiency. Weinberg and Schulman found that the objective was achieved at the expense of the other measure. The group with the efficiency goal minimized execution time but required a greater number of runs, while the fast development group produced the program more quickly but execution time was much longer.

In their second experiment, Weinberg and Schulman (74) studied the performance of six three-member groups. Each group was assigned a primary goal of minimum core used, minimum execution time, output readability, program readability, minimum statements, or minimum programming hours; each group chose a secondary goal from the same list of six goals. Performance was measured directly for all attributes except readability, which was judged by ranking of solutions by experts.

Weinberg and Schulman (74) report a clear influence of the primary goals on performance. Some goals were found to be incompatible, such as execution efficiency and program readability; in particular, core and execution optimization goals were found to conflict highly with other goals. Programming groups maximized performance on the primary goal attribute when faced with incompatible multiple goals. Note that programmers can make tradeoff decisions when given explicity ranked goals.

Programming Tools and Methods

The tools and methods used on a software development project also clearly affect performance. A summary of these factors is shown in Table 5. Both Brooks (74) and Oliver (78) compared the effect of language level on productivity, finding that productivity, measured as lines of code produced per man month, remained constant regardless of language level. From a psychological point of view, this is reasonable if each line of code or instruction is considred a conceptual unit or chunk. Despite the fact that line-by-line productivity is relatively constant, Brooks estimates that effective productivity increases by 5 times when higher level languages such as FORTRAN are used instead of assembler language. This is because fewer lines of code are required in a higher level language to accomplish the same process.

The use of modern programming practices such as structured programming techniques also has been found to dramatically increase productivity. Based on project audits at IBM, Baker (75) estimates that full implementation of structured programming techniques can yield a 50% increase in productivity, measured in bytes of code produced per man month. Full implementation of structured techniques includes use of the Development Support Librarian, Top-Down Development, Structured Coding, and Chief Programmer Teams. In two comparable aerospace projects on which the experience level of the personnel was equivalent, the use of the Development Support Librarian alone increased overall productivity by 50%. Baker (75) also compared productivity rates of two other aerospace projects. On one, the Development Support Librarian was the only technique used; on the other, the Development Support Librarian, Top-Down Design and Structured Coding were all employed. The personnel on the second project produced roughly twice as many bytes of code per man month.

While clearly IBM's experience with structured programming techniques has yielded impressive productivity gains, it may not be this specific set of techniques that is responsible. It may be that any structuring and formalizing of the methods and approach to software development would yield significant improvements in productivity.

Programmer Factors

Programmer experience, especially experience in the specific application area, is another factor which has been found to increase productivity (Scott and Simmons, 1974). A particularly interesting result was found by Youngs (74), who investigated the effect of programmer experience on error frequency. Novice programmers were compared with "professionals," defined by Youngs as persons who had earned money programming. It is certainly reasonable to assume that the level of expertise of the "professionals" differed considerably.

Youngs' (74) results are especially interesting because they are somewhat counterintuitive; professional programmers committed more logical errors than novices. Logical errors were defined as errors in the algorithm to solve the problem which produced incorrect program output. Youngs also found, as might be expected, that professional programmers made fewer total errors and required fewer runs to achieve a correct program.

Multifactor Research

Scott (73), in addition to his simulation research on programming group organization, used multiple regression techniques to determine influences on project-level productivity. Two large software development data bases, from SDC and PRC, were used. Total lines of object code produced over the total man months of the project was the dependent variable; the order in which the eight most significant independent variables were entered into the equation is shown in Table 6. This order represents the proportion of variance of the dependent variable, productivity, which was accounted for by the independent variable. Thus, the most significant factor was frequency of operation of program, which is related to Brooks' levels of complexity. The plus and minus symbols to the right of the independent variables indicate their relationship to the dependent variable. A "+ +" indicates that an increase in level of the independent variable causes an increase in level of the dependent variable, productivity. In general, this list of factors influencing performance is consistent with the factors summarized above.

Additional confirmation of the influence of these factors is found in another research study by Scott and Simmons (74), who used the Delphi technique to achieve expert consensus on factors influencing project-level productivity. After two rounds, subjective agreement was achieved on factors influencing implemented object instructions per man month. On a fifteen point scale, from -7 to +7, the factors with the highest median ratings, indicating greatest positive effect on productivity, are shown in Table 7. These factors are much the same as those already addressed, with the exception of the addition of task allocation factors. Unfortunately, variables having a highly negative effect on productivity were not reported.

Implications and Conclusions

The factors identified which influence performance were summarized in Table 1. Clearly, some of the factors, especially project and environment factors, are beyond the control of project management. However, prescriptive conclusions may be generated from an examination of factors that may be manipulated at the management level. These conclusions and recommedations are shown in Table 8.

First, always choose the highest level language that will satisfy the requirements of the problem. The use of a high level language not only increases productivity, but enhances product maintainability and transferability.

In selecting a programming group organization, communication bottlenecks should be minimized by using some structure other than the traditional team organization. The subteam structure used by Scott avoids communication bottlenecks and also seems to minimize the communication requirements placed on each team member. Also, to maximize group productivity, a highly productive individual should be assigned as team member, not team leader.

Individual task assignments should minimize the routine interaction and communication between team members. When possible, task assignments should consist of functionally independent modules. In addition, of course, task assignments should be explicitly documented and communicated to the individual.

At the project management level, the explicit statement of goals and ranking of priorities will maximize performance on the primary goal and allow for reasonable product performance tradeoffs to be made.

Well-defined practices such as structured programming techniques should be implemented to improve productivity. Furthermore, the project manager should avoid the impulse to overload a project with personnel. Brooks Law (74) still holds: in addition to reaching a point of diminishing returns with an increase in personnel, productivity on a late project may be impaired. Finally, design and code walkthroughs should be implemented in a non-threatening manner, without representatives from management, in order to detect logical and other errors which may elude the individual programmer.

References

Baker, F.T. Structured programming in a production programming environment. In Proceedings, International Conference on Reliable Software. SIGPLAN Notices, 1975, 10 (6), 172-185.

Brooks, F.P., Jr. The mythical man-month. Datamation, 1974, 20 (12), 44-52.

Johnson, J.R. A working measure of productivity. Datamation, 1977, 23 (2), 106-112.

Oliver, P. Examining programming costs. Computer Decisions, 1978, 10 (4), 50-52.

Sackman, H., Erikson, W.J. and Grant, E.E. Exploratory experimental studies comparing on-line and off-line programming performance. Communications of the ACM, 1968, 11, 3-11.

Scott, R.F. A computer programmer productivity prediction model. Unpublished doctoral dissertation, Texas A&M University, College Station, TX, 1973.

Scott, R.F. and Simmons, D.B. Programmer productivity and the Delphi technique. Datamation, 1974, 20 (5), 71-73.

Scott, R.F. and Simmons, D.B. Predicting programming group productivity--a communications model. IEEE Transactions on Software Engineering, 1975, SE-1, 411-413.

Weinberg, G.M. The Psychology of Computer Programming. New York: Van Nostrand Reinhold Company, 1971.

Weinberg, G.M. and Schulman, E.L. Goals and performance in computer programming. Human Factors, 1974, 16, 70-77.

Youngs, E.A. Human errors in programming. International Journal of Man-Machine Studies, 1974, 6, 361-376.

Zelkowitz, M.V. Perspectives on software engineering. ACM Computing Surveys, 1978, 10, 197-216.

TABLE 1
FACTORS AFFECTING PERFORMANCE

PROJECT FACTORS

Program Type

Program Complexity

Project Size

Frequency of Operation

Interaction Between System Components

Elapsed Time for Development

Percent I/O Instructions

ENVIRONMENT

Timesharing vs Batch

Turnaround Time

TOOLS AND METHODS

Programming Practices/Standards

Language

Tools and Utilities

MANAGEMENT FACTORS

Statement of Objectives

Number Personnel Assigned

Organization of Teams/Groups

Required Communication

Documentation Quality

Task Assignment

PERSONNEL

Experience in Data Processing

Experience in Application Area

TABLE 2

Project Factors

Factor	Researcher	Type of Research	Performance Measure	Relationship of Factor to Performance
Program type: control system application	Zelkowitz (78)	Summary	LOC/year	-Control programs written at 600 LOC/year System programs at 2000 LOC/year Application programs at 6000 LOC/year
Program type	Brooks (74)	Summary	implemented object instructions/year	-3-fold increase in productivity for system program over control program -9-fold increase for application program over control
Program complexity: program programming system programming product programming system product (see text)	Brooks (74)	Summary	Cost factor ≈ productivity	-Program cost factor = 1 Programming system = 3 Programming product = 6 Programming system product = 9
Project size	Johnson (77)	Project Audits	debugged, documented lines of source code/year	-Productivity rate on small projects 3x more than on large projects
Interaction among system components	Brooks (74)	Summary	implemented object instructions/year	-High degree of interaction reduces productivity.

TABLE 3

Environment Factors

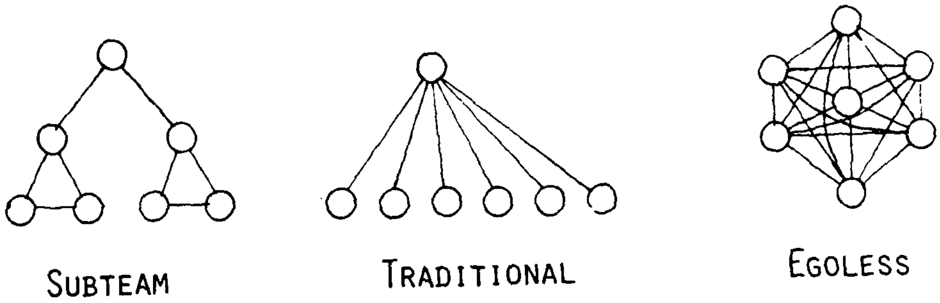
Factor	Researcher	Type of Research	Performance Measure	Relationship of Factor to Performance
Timesharing vs. batch	Sackman, Erickson and Grant (68)	Controlled Experiment	CPU time debugging hours coding hours execution time program size (machine instructions)	-Large individual differences: 28:1 debug time, 26:1 coding time -Significantly lower debug time with timesharing when scores transformed and coding skill covaried out -Suggest turnaround time is key factor
Turnaround time	Oliver (78)	Observation	productivity (undefined)	-With faster turnaround time productivity increases

TABLE 4

Management Factors

Factor	Researcher	Type of Research	Performance Measure	Relationship of Factor to Performance
Programming team organization traditional egoless subteam (see Fig. 1)	Scott (73)	Modeling/Simulation	Scaled to lines of implemented object code/month	-Egoless and subteam structures had significantly higher group productivity than traditional.
Role of highly productive individual	Scott (73)	Modeling/Simulation	Scaled to lines of implemented object code/month	-In traditional team, potential of highly productive individual reduced if team chief.
Team Size	Scott (73)	Modeling/Simulation	Scaled to lines of implemented object code/month	-In traditional team, group productivity peaks at team size of twelve members.
Project Goals	Weinberg and Schulman (74)	Controlled Experiment	Program execution time # runs to completion	-With explicit goal, performance was maximized on goal criterion.
Ranked project goals	Weinberg and Schulman (74)	Controlled Experiment	Execution time Core used Programming hours # of program statements Output readability Program readability	-Programming groups maximized performance on primary goal -Clear influence of primary and secondary goals on performance -Some multiple goals incompatible. (see text)

FIGURE 1



SUBTEAM TRADITIONAL EGOLESS

EXPERIMENTAL TEAM STRUCTURES USED BY SCOTT (73) AND SCOTT AND SIMMONS (75)

TABLE 5

Tools and Methods

Factor	Researcher	Type of Research	Performance Measure	Relationship of Factor to Performance
Language level: Assembler COBOL FORTRAN PL/I	Brooks (74)	Summary	lines debugged code/man month	-LOC constant regardless of language level. -Use of high-level language yields effective productivity increases of 5x over machine language
Language level	Oliver (78)	Observation	productivity (undefined)	-Productivitv constant reeardless of language.
Structured programming techniques: -Development Support Librarian (DSL) -Top Down Design (TDD) -Structured Coding (SC)	Baker (75)	Project Audits	bytes/man month	-Use of DSL vs. no tools increased productivity by 50% -Use of DSL, TDD and SC vs. DSL alone increased productivity by 100% -Estimated 50% increase in productivity with full implementation of structured programming techniques

TABLE 6

Order of Independent Variables Entered into Regression Equation

1.	Frequency of operation of program	+	-
2.	Programmers assigned	+	-
3.	Elapsed time required for development	+	-
4.	Use of low level language	+	-
5.	# Analysts assigned	+	-
6.	% of I/O instructions	+	+
7.	Average programmer experience	+	+
8.	Complexity of application	+	-
9.	Response time required of program (from batch to real-time)	+	+

Adapted from Scott and Simmons (75)

TABLE 7

Factors with Greatest Influence on Productivity

		MEDIAN RATING
1.	Quality of external documentation (prior to task assignment)	(6)
2.	Programming language	(5)
3.	Availability of programming tools (e.g., utilities, traces & dumps)	(5)
4.	Programmer experience in data processing	(5)
5.	Programmer experience in functional area	(5)
6.	Effectiveness of project communications (completeness of task assignment)	(5)
7.	Independent modules for task assignment	(4)
8.	Use of well-defined programming practices	(4)

Adapted from Scott and Simmons (74)

TABLE 8

Conclusions

Language

- use highest level possible

Team Structure

- use subteam or egoless structure
- assign highly productive individual as team member, not chief.

Task Assignments

- Divide tasks to minimize intercommunication
- assign functionally independent modules
- explicitly document and communicate assignment

Management

- explicitly state goals and rank priorities
- avoid overloading project with personnel
- implement design and code walkthroughs
- use well-defined practices and methods

Human Factors in Query Language

Lawrence M. Potash

Army Research Institute

Development of interactive query language systems that approach use of natural language in flexibility and power and employment of such systems by the Army makes a review of human factors considerations in development of these systems highly desirable. Human Factors considerations in development of query language systems were discussed under the topics:

a. Query language

b. Symbols or vocabulary used in query language

c. Supportive, pacing and general features of man/computer dialogue

d. Types of users employing the system.

Some human factors research undertaken at USARI relevant to these topics had also been briefly discussed.

HUMAN FACTORS IN QUERY LANGUAGE

Lawrence M. Potash

U.S. Army Research Institute for the
Behavioral and Social Sciences
Alexandria, Virginia

The development of interactive query language systems that approach use of natural languages in flexibility and power and the employment of such systems by the Army (for example GIMII used in ASSIST) make a review of relevant literature desirable.

This paper is concerned with query "languages" that have syntax that is more than an elementary "fill in the blank" or picking out appropriate terms from a hierarchical list. Examples of query statements in such "syntactical" languages are shown below:

```
                 SQUARE:              EMP                    ("50")
FORMAL                      NAME               DEPTNO
LANGUAGES

                 GIMII:  FROM EMP WITH DEPTNO EQ "50" LIST NAME #

NATURAL
LANGUAGE         ENGLISH:  Find the names of employees in Department # 50.
```

A query language system is more than just the basic query language. Adequate description must also include supportive features ("help" , clarification Dialogue, error feedback, etc.), pacing devices, (confirmatory signals, attention signals, etc.), and the user population for which the system is intended.

This literature review synthesizes experiments, theoretical and descriptive literature relating to query language. Relevant literature is summarized in terms of a) syntax, b) symbols or terms used in query language, c) supportive, pacing, and general features of the man-computer dialogue, d) types of users employing the system. The literature review on which this paper is based is nearing completion and is quite long. Rather than trying to condense the entire review into the written equivalent of a 20 minute presentation, I am listing some of the more major conclusions and suggested areas for future research in terms of the four dimensions previously described. Before listing these conclusions, one generally applicable, perhaps somewhat disappointing, cautionary note is that MOST CONCLUSIONS OR ASSERTIONS THAT ARE FOUND IN THE LITERATURE ARE THE RESULT OF USER EXPERIENCE AND/OR "REASONABLE" OR LOGICAL EXTENSION OF SUCH EXPERIENCE.

Currently, a large body of research literature which could serve to provide Human Factors guidelines for query language development DOES NOT EXIST. The remainder of the paper cites some of the more important conclusions derived from the literature survey.

QUERY LANGUAGE SYNTAX

Some Conclusions

o Advantages of natural language syntax ("english") are limited training requirements and flexibility. Disadvantages of natural language are imprecision and difficulty of implementation.

o Disadvantages of using natural language are diminished when the domain of subject matter is highly constrained.

o System "comprehension" of query language ranges through key word recognition systems such as ELIZA (which can be made to give the appearance of english comprehension) through formal query languages (as per example), which are relatively limited in range of procedures, syntax, "meaning", and their domain of competence, to truly generative comprehension in which the system can analyze a natural language input using internal knowledge of the subject domain and reasoning capability to generate its own questions, problems or solutions. Only a few generative systems exist and all have a very restricted domain of competence.

o In future systems using voice input for natural language, the users should be given incentives for being precise and concise.

o Use of logical and arithmatic operators may result in high error rates (indicated in some but not all research literature).

Some Suggestions For Future Work

o Cost benefits analysis including experimentation to delineate conditions for advantageous use of natural language.

o Investigate use of logical and arithmatic operators embedded in different query languages and employed by different user populations.

o Comparison between different approaches to formal query language such as QUERY BY EXAMPLE, SEQUEL, SQUARE, GIMII under variety of conditions and user parameters.

o Assessment of an interesting alternative to truly generative systems, use of systems that employ task related information to aid the user.

SYMBOLS OR TERMS USED IN QUERY LANGUAGE

Some Conclusions

- Controlled vocabulary can facilitate searches once vocabulary has been learned.

- "Inverted glossary" with system displaying legal control terms after user inputs terms can aleviate learning requirement of controlled vocabulary.

- User codes should suggest what they represent (i.e., letters making up acronyms rather than arbitrarily assigned letters or numbers, etc.).

- Experimental work indicates that simple truncation is an effective abbreviation technique but much more research needs to be done.

- For natural language, a relatively small vocabulary is probably satisfactory for most purposes when the subject matter handled by the system is not too broad.

Some Suggestions For Future Work

- Research on category definition and selection of most effective labels or retrieval terms.

- Maximizing abbreviations to best represent terms they stand for.

- Effects of restricted vocabulary on use of natural language in realistic man-computer dialogue settings.

SUPPORTIVE, PACING, AND GENERAL FEATURES OF MAN COMPUTER DIALOGUE.

Some Conclusions

- Tolerance for delay in system response is related to perceived difficulty of problem (10 sec for relatively small computations to 10 min for long problems).

- Tolerance of delay is enhanced by pacing devices, i.e. (confirmatory signals, attentional signals, cueing signals, status displays).

- Error - control program advantageous (entry preparation display, editing facilities, variable spelling approximations, flexible error description feedback, system error monitoring).

- Error messages should not

    1) be humoureous or overly friendly (it "wears thin").

    2) Use wording that implies fault on part of user.

o Error messages should communicate

1) where error occured

2) what error is

3) ways to recover from error (or where to find relevant information.

o CRT displays permit more rapid communication, are less noisy, allow user errors to be corrected more easily than teletypewriters.

Some Suggestions For Future Work

o Work could be undertaken to determine how support requirements differ for different types of query language such as natural versus the more popular types of formal language.

TYPES OF USERS EMPLOYING SYSTEM

Some Conclusions

o Distinction between dedicated operator, casual operator, and intermediary operator.

o When casual users employed

1) system should not be difficult to learn or have many new operations

2) feedback should tell user exactly what to do

3) system should not place short time limits on user's response (pressure effect)

4) have terminal in private rather than public area (fishbowl effect).

o When casual and dedicated users employ system it should be flexible (i.e., layers of language, detailed error feedback for casual user, abbreviated feedback for dedicated user, etc.).

o Where a keyboard is used by unskilled typist, function keys, minimal character recognition, etc. may be helpful or intermediary operator with typing skills could be employed.

o Users with programming background may use less English like formal languages more accurately.

Some Suggestions For Future Work

o Field concerning interaction of system with user characteristics relatively "wide open" for empirical study as contrasted with induction from "previous experience" or deduction from "reasonable assumptions".

Issues in Human/Computer Interaction

Raymond C. Sidorsky

Army Research Institute

Human Factors needs help from other disciplines if it is to rise above ad hoc "solutions" as each man/computer interface is developed. Other disciplines are in the same boat. Means must be devised to enable interdisciplinary discourse within a context that is broader than the specific system for which the team is assembled. Modes and mechanisms of man/computer interaction that transcend specific systems must be identified and characterized by interdisciplinary Working Groups. Macros, tasks modularization, logical operators, retroactive error recovery and other candidate topics have been discussed.

INTERDISCIPLINARY ISSUES IN THE DESIGN OF EFFECTIVE HUMAN-COMPUTER INTERFACES

Raymond C. Sidorsky
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

The thrust of my remarks is to make a plea for greater interdisciplinary interaction in the system design process. I shall also discuss a possible mechanism for increasing such interaction within ISRAD.

The Human Factors community needs help from other disciplines if it is to rise above ad hoc solutions as each man-computer interface is developed. In almost every case human factors specialists are faced with trying to make the best compromise within a very narrow range of options left open to them. I realize that the design of any complex system requires a large number of compromises and tradeoffs with respect to such factors as physical layout, electronics, software, user requirements, system architecture and so forth. However, the other disciplines can always hope that whatever the problems, the "user" will be able to cope with the situation and make the system work at a level which, if not ideal, is at least sufficient to justify the development of the system. When the system is fielded we in the human factors business usually find ourselves trying to make the best of a bad situation. After we have done our thing, we are often left with the feeling that although we have helped alleviate the immediate problem, we have not developed a body of knowledge that can guide us in developing future systems. Each system turns out to be a special case. Slide #1 illustrates this situation. Each of the three Human Factors experiments shown here was concerned with a different kind of user problem encountered in the operation of an automated tactical data processing system such as TOS, TACFIRE, ASSIST, etc.

The first deals with the problem of the restricted vocabulary and nomenclature involved in human-computer discourse. Nystrom and Gividen at the ARI Ft Hood Field Unit observed that TOS users were consistently misclassifying tactical data messages. A message classification coding schema based on a letter-letter-number schema appeared to be the proximate cause. An analysis of the structure and content of TOS messages followed by empirical performance measurements led them to device a four letter coding schema. The new schema significantly reduced the number of misclassified messages and was more satisfying to the users.

The process of transforming raw information, e.g., spot reports, into computer acceptable form is another recurrent problem in tactical data system operations. Strub compared operator performance under four conditions, viz., on-line versus off-line entry and verified (by a second operator) versus non-verified message composition. Performance measures of speed and accuracy indicated that the on-line verified procedure was significantly superior to the other methods.

Finally, Fields, et al. evaluated four methods of inputting data. Speed and accuracy of performance in using standard typing procedure was compared with typing plus error correction, typing plus automatic completion of entries and light pen selection of entries from CRT displayed "menus." Menu selection proved to be the most effective.

Each of these studies is a good example of a well designed, carefully controlled empirical evaluation of alternative configurations at the human-computer interface. They provided valuable information to help resolve critical problems in the design or operation of particular systems. But only under exceptional circumstances will it be feasible to extrapolate these findings to future systems. The problems have been presented to the human factors specialists in a way that makes the development of general principle and/or guidelines extremely difficult if not impossible.

My purpose is not to try to pass the buck or to deliver an apologia for any real or imagined failings of Human Factors. The low level of user performance obtained with many systems is not the result of a conspiracy to give human factors types a hard time. Instead it appears to be the result of an inability of various team members to communicate and interact with each other at a level that transcends the specific requirements of the system for which the team has assembled. The basic problem exists for all members of the design team, not just human factors. All of the team members appears to view the situation as one in which their freedom of action is unduly constrained by the inflexible, unalterable requirements of other team members. But I think these seemingly unalterable "requirements" may be more apparent than real. Their origins may lie in the natural defensiveness of the members of each group striving to minimize the complexity of their problem and/or in the erroneous assumption that some desirable feature is well beyond the state-of-the-art of a companion technology. That is, they don't ask for a new capability because they assume it couldn't be delivered. Whatever the cause, in many cases the result is a design decision based on too little information about its effects and implications. Now everybody knows that "better communication" is needed in almost every sphere of human activity. So what else is new?? I intend my remarks to be more than a mere exhortation. So I will try to describe some actions that could be taken by members of ISRAD to help all of us to achieve more effective system design and operation.

The method I propose is an adaptation of the Delphi Procedure that has been used successfully for technological forecasting. I feel the Delphi technique may provide a means for developing a mode of discourse that will enable more constructive dialogue among system design team members. Two kinds of Delphi-type working groups are needed. The first stage Working Group (or Groups) would be tasked to define or identify system operations or processes for further analysis. To be useful, the operations or procedures selected would have to be general enough to transcend specific system constraints but at the same time not so general or vague that they can only be discussed at a philosophical level.

Second stage Working Groups would then be set up to pursue the topics defined by the first stage Working Group(s). The objective of these second stage groups would be to try to determine--using iteration, feedback, anonymity and other Delphi procedures-the practical limits that can be achieved in utilizing or improving various processes or procedures associated with information processing systems. An open, frank discussion of the needs, limits and capabilities of each component technology--in a non-threatening Delphi atmosphere--would go a long way toward dispelling the many myths, fears and misconceptions that frequently impede the system design process.

The following are some possible Working Group topics. These topics are offered not with firm conviction as to their utility but merely as illustrations of the approximate level of abstraction of the variables to be analyzed. All have a human factors focus but are contingent upon an integrated effort of all members of the system design team for their resolution. I'd like to discuss each of these topics briefly.

DATA ENTRY MACROS.

The first item, Data Entry Macros, are to be distinguished from Procedural Macros utilized by programmers. The term macro is used here to refer to the sort of shorthand notation used to represent groups of computer instructions that comprise a sequence of logically interrelated operations upon some variables of interest to a system user. For example, as shown in the next slide (Figure 3) a typical request from a commanding officer to the G-2 staff might lead to a query statement requesting the computer to:

"access the ENSIT file and print the time, location and direction of movement of all enemy patrols observed in area XYZ."

Using macros, the operator need only enter:

M7, patrols, XYZ

that is, only three entries versus 12 in the original statement.

Thus, the operator need only enter two or three items of information to supplement the large number of entries made automatically by the computer. This would inevitably reduce input and logical errors, increase the speed of data entry and retrieval and promote more effective communication between the system user (e.g., G-3, G-2) and the interface (keyboard) operator.

Macros are potentially a great boon for the system user, especially for the entry or retrieval of recurrent messages or queries. Indeed, the macro approach may be the key to a conceptual breaktrhough in the barrier between the user and the data base by allowing the user to tailor the input/output process to the specific needs of the current situation. However, much of the potential value of the macro approach will be lost if macros are not developed in a manner compatible with the behavioral characteristics of the user population. Experience has shown that the cognitive and mnemonic burden imposed on the TOC staff and the terminal operators by poorly organized data base structures is very high. The haphazard or casual use of macros will not prevent high error rates, low productivity and user antagonism. The full benefit of data entry macros will be obtained only when all members of the system design team are aware of their properties, operation, advantages and limitations, etc. The team members involved include those responsible for defining user requirements as well as software, hardware, system architecture, human factors and other technical specialists.

The implications of a number of the general aspects of macro design and use must be carefully analyzed if the maximum benefits of macros are to be realized. This includes consideration of such questions as:

Universal versus locally generated macros. A library of macros to accomplish common, more-or-less universal entry and retrieval operations must be defined and standardized. However, caution is needed to insure that over-standardization does not inhibit the ability of users to devise individualized macros to suit their local circumstances. The factors involved in the trade-offs between standardization and flexibility require careful, coordinated analysis by all system design team members.

Macros within macros. As experience is gained with macros, it will become apparent that basic macros can be aggregated to create higher order macros. The development of higher order programming languages (PL-1, Fortran, etc.) is perhaps an appropriate analogy. Since higher order macros may provide the key to minimally constrained human-computer interaction, care must be exercised to avoid premature standardization on processes or procedures that will inhibit an optimum approach to higher order macros.

Non-stationary population. One obvious fact of life facing us in the development of macros is that the pool of potential users of any system contains individuals who vary widely in skill and/or experience. Furthermore, a given individual changes from novice to expert as a function of practice. The developers of data entry macros-basic as well as higher order--must take into account these skill/experience differences and provide sets of macros that will accommodate the entire range of user skill/experience levels. In other words, any set of macros currently under development that does not provide for simultaneous use by novices and experts is seriously flawed.

LOGIC AND LOGICAL OPERATORS.

Boolean algebra expresses truth in a particular way. It enables us to express and keep track of the relationship between sets and variables in a way that is very useful in the design and operation of switching circuits and computers. Unfortunately, it doesn't correspond particularly well with the way people analyze and keep track of things. With special effort, some people can be trained to view the world in Boolean terms. But only a small number of people can do it and the effort involved is enormous. Some way has to be found to make computers understand people rather than forcing people to adapt to a non-conventional mode of thought and discourse. The use of data entry macros discussed earlier may be one mechanism. For example, although the term Macro 7A, xx, yy might imply different logical operations to a human and a computer, the result produced by the computer will satisfy the request of the user. Here the situation is somewhat analogous to the use of ideographics for communication by Chinese. The speech of a Cantonese is unintelligible to a resident of Peking and vice versa. However, they can communicate with ease via the mutually understood written characters. In any event, the problem of logic and logical operators cannot be solved by specialists in the relevant technologies working in isolation. An integrated, concerted effort is needed.

TASK MODULARIZATION.

Even a cursory observation shows that many of the procedures employed in the operation of various automated systems are basically similar. Yet from the operator's perspective each system is a new situation with little carryover or transfer from the other systems. Take terminology, for instance. One gets the impression that system designers, like the Weather Bureau naming hurricanes, use a different combination of assigned names for each new system. "Get" is alternated with retrieve, fetch, obtain, bring, call, etc. Similar clusters of synonyms are used for operations such as "store" or "list" or "transfer" or other common operations. The number of combinations of the half dozen or so synonyms for the dozen or so most frequently used operations is in the billions. We seem to be determined to use every one of them. It may be too early to establish absolute "standards" for terminology but a measure of consistency would go a long way toward making computer systems more approachable to Army users.

A similar situation exists with respect to message formats. Every formatted message contains a common set of elements and operations, e.g., orginator, addressee, data file designators, operating instructions, subject(s), actions taken by or affecting the subject(s), administrative details (security, precedence, serialization), etc. Arranging the messages in a consistent sequence would greatly increase the speed and convenience with which military users/operators could interact with a data processing system upon first encounter.

Consistent terminology and message formats are somwhat elementary examples of task modularization. The concept can be extended to encompass dynamic processes associated with human-computer operations as well. For example, the underlying processes involved in setting up a data file or of retrieving a sub-set of the information in a data file are functionally similar regardless of the subject matter contained in the file. Yet each existing system requires a different configuration of operations and procedures for executing such functions as data entry, file set-up, data retrieval, data transfer, etc.

Although the use of standardized "task modules" might reduce system flexibility somewhat, the loss would probably be more than compensated for though reduced operator error and training time. However, a valid determination of the net trade-off of all relevant costs and benefits would require inputs from all of the disciplines involved in the design and operation of computerized information processing systems.

This brings me to the main point of my remarks, i.e., what has all this to do with ISRAD? The notion of a Delphi procedure for coordination of technological know-how has some obvious plausibility when considered in the abstract. The virtue of developing various sorts of "standards" and "modules" is universally recognized along with motherhood, apple pie and Old Glory. However, I believe that the ISRAD group has some characteristics that make it well suited as a medium through which various trans-systems modes and mechanisms of operation can be identified and standardized.

These characteristics include the presence of a critical mass of personnel in all of the relevant technologies. There are probably 10 or more engineers and scientists back at the office for each of the 200 or so people here today. Not many organizations can muster this much talent across such a wide range of disciplines.

ISRAD is mission oriented. That is, although we represent many different technologies and organizations, we all share a common-goal; to design and produce information systems that will help the US Army accomplish its overall mission. This esprit de corps is a valuable asset in interagency and interdisiciplinary cooperation.

Non-profit motivation. Because we are not concerned with the commercial implications of data processing systems we are not constrained by consideration of industrial secrets, market protection, product obsolescence and the like. We can exchange information freely and base decisions on criteria that serve long range national goals.

The ISRAD group encompasses a wide spectrum of technologies. The entire gamut of system design specialties, e.g., military (user) requirements analysts, electronic engineers, computer designers, programmers, human factors, cost analysts, production engineers, etc., are available within ISRAD or are readily accessible to it.

For the most part, ISRAD agencies deal with the application of developed technology. The Army's primary concern is to field systems whose technological soundness and reliability are proven. Thus, we need not be preoccupied with making sure that our "standards," macros, task modules, etc., can accommodate esoteric or untested techniques being investigated at the laboratory level.

All of these characteristics taken together make ISRAD an ideal group to undertake an interdisciplinary approach to the functional standardization of terminology, data entry macros, logic systems, task modularization and other modes and mechanisms of human-computer interaction.

REFERENCES

Nystrom, C.O. and Gividen, G.M. Ease of Learning Alternative TOS Message Reference Codes. Technical Paper No. 326, US Army Research Institute, Alexandria, Va, September 1978.

Strub, M.H. Evaluation of Man-Computer Input Techniques for Military Information Systems. Technical Research Note No. 226, US Army Research Institute, Alexandria, Va., May 1971. (AD 730 315).

Fields, A.F., Maisano, R.E. and Marshall, C.F. A Comparative Analysis of Methods for Tactical Data Inputting. Technical Paper No. 327, US Army Research Institute, Alexandria, Va., September 1978.

REQUIREMENTS I

Dr. Edward Lieblein
CENTACS

REQUIREMENTS I

SESSION CHAIRPERSON: Dr. Edward Lieblein

CENTACS

SESSION SUMMARY

It has become clear that inadequate approaches to software/system requirements development have contributed more to the high cost and poor performance of software than any other area. This is especially true in the domain of embedded computer systems where the software is not readily separable from the hardware. Inadequate specification of initial requirements may be carried through several levels of "requirements engineering" before they are detected, and in many cases, such problems are not detected until the project has reached the programming, integration and test, or even the operational phase. The relative cost to correct a specification or design error increases significantly when such errors are discovered in later phases of the project.

This session explored the complex area of software/system requirements from several viewpoints. The first paper described experiences with respect to application of the Software Requirements Engineering Methodology (SREM), a computer-aided approach to definition and analysis applicable to real-time systems. The second paper discussed a hierarchical decomposition methodology that takes requirements through various stages including formal specifications of requirements "modules". The decomposition of requirements for a secure real-time tactical executive was described as an illustration of the approach. The third paper addressed the issue of software requirements from the viewpoint of the project manager who is concerned with the specification of software in the Request for Proposal for an embedded computer system and management control for software development throughout the contractual effort.

Computer-Aided Requirements Generation
An Evaluation

Carl G. Davis

BMD Advanced Technology Center

The application of the computer as an analysis tool in the requirements definition phase of system development has been demonstrated through the application of the Software Requirements Engineering Methodology (SREM) to a wide variety of projects. This paper discussed conclusions drawn from experiences with SREM and suggested research directions to further enhance the requirements definition process.

COMPUTER-AIDED REQUIREMENTS GENERATION--AN EVALUATION

Carl G. Davis
Ballistic Missile Defense Advanced Technology Center
Huntsville, Alabama

Introduction

The Software Requirements Engineering Methodology (SREM) was developed as an integral part of an overall software development approach, entitled the Software Development System (SDS) [1]. SREM was developed to significantly improve the capability to develop requirements for a data processing subsystem when system-level definition had been given. This system was designed for problems inherent in software development for Ballistic Missile Defense (BMD) systems and was sponsored by the Ballistic Missile Defense Advanced Technology Center (BMDATC), Huntsville, Alabama.

SREM was developed during the period of 1974 to 1977 and evolved through experiences gained and requirements derived from application to a successive set of complex problems. The approach has thus been verified through experience during the development process. This paper will describe the experiences gained in evaluation of SREM and discuss the evolving nature of requirements engineering at BMDATC.

SREM Description

SREM was developed as an approach to aid in the generation of BMD data processing subsystem requirements. It consists of a combination of languages, analysis tools, and procedures designed to allow effective statement of requirements and to eliminate or reduce known error sources [2,3] (Figure 1). Requirements are stated in the machine analyzable Requirements Statement Language (RSL). The requirements may be stated interactively or in a batch mode and are checked for consistency with previously entered data and for completeness of description via automated analyzers. Structure is provided through R-NETS, which form a path-oriented description of the system. The requirements description is stored in a relational data base, which is accessed through a flexible retrieval system. Computer-aided simulation generation allows rapid evaluation of the dynamic nature of the stated requirements. Upon the completion of static and dynamic validation, the requirements are documented using automated aids. Early structuring and analysis provide for rapid feedback to the system designer.

SREM also includes procedures, steps, rules, etc., for the development of requirements. Cost and scheduling models have been developed that enhance the ability to estimate the impact of the requirements phase of software development. The control and management of the methodology are interwoven and based upon the defined tools and techniques, allowing greater efficiency and accuracy in management information.

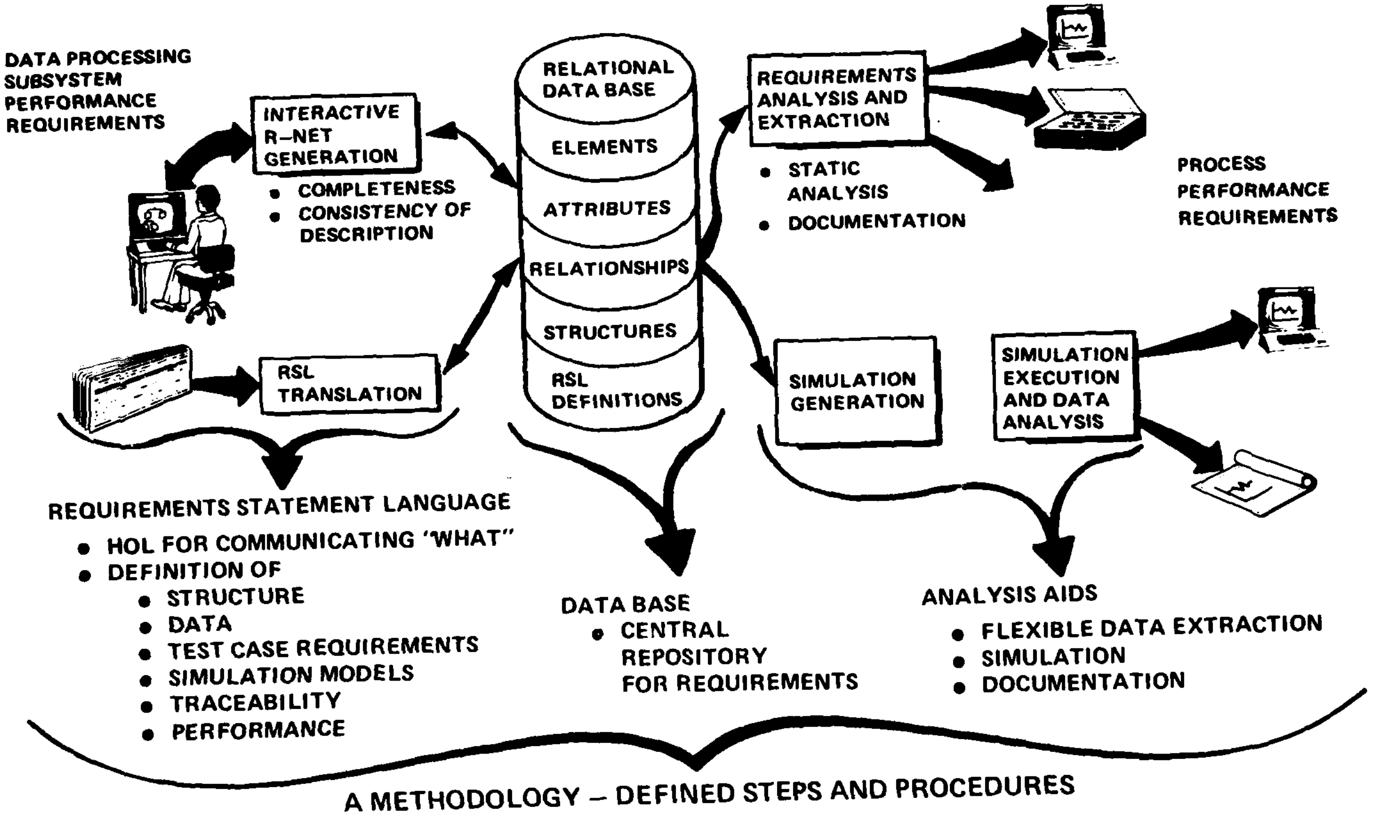

DATA PROCESSING SUBSYSTEM PERFORMANCE REQUIREMENTS
INTERACTIVE R–NET GENERATION
• COMPLETENESS
• CONSISTENCY OF DESCRIPTION
RSL TRANSLATION
RELATIONAL DATA BASE
ELEMENTS
ATTRIBUTES
RELATIONSHIPS
STRUCTURES
RSL DEFINITIONS
REQUIREMENTS ANALYSIS AND EXTRACTION
• STATIC ANALYSIS
• DOCUMENTATION
PROCESS PERFORMANCE REQUIREMENTS
SIMULATION GENERATION
SIMULATION EXECUTION AND DATA ANALYSIS
REQUIREMENTS STATEMENT LANGUAGE
• HOL FOR COMMUNICATING 'WHAT"
• DEFINITION OF
• STRUCTURE
• DATA
• TEST CASE REQUIREMENTS
• SIMULATION MODELS
• TRACEABILITY
• PERFORMANCE
DATA BASE
• CENTRAL REPOSITORY FOR REQUIREMENTS
ANALYSIS AIDS
• FLEXIBLE DATA EXTRACTION
• SIMULATION
• DOCUMENTATION
A METHODOLOGY – DEFINED STEPS AND PROCEDURES

Figure 1. The Approach—Requirements Engineering

Requirements Structure

The specification of the structure of processing steps in RSL is through the element R-NET [4]. Each R-NET details the response of the system to particular stimuli through defining the sequence of ALPHAs (processing steps) to be followed to generate changes in system state and responses to the environment. When all of the required steps are completed, the R-NET processing terminates. The sequence of ALPHAs is specified by giving a graph model of the sequence in a structure declaration associated with the R-NET. A sample R-NET showing both graphical and textual form is shown in Figure 2. The flow structure of an R-NET consists of nodes and the arcs that join them. Five types of primitive nodes may be placed at any point on the structure (e.g., and/or nodes).

Requirements Statement Language (RSL)

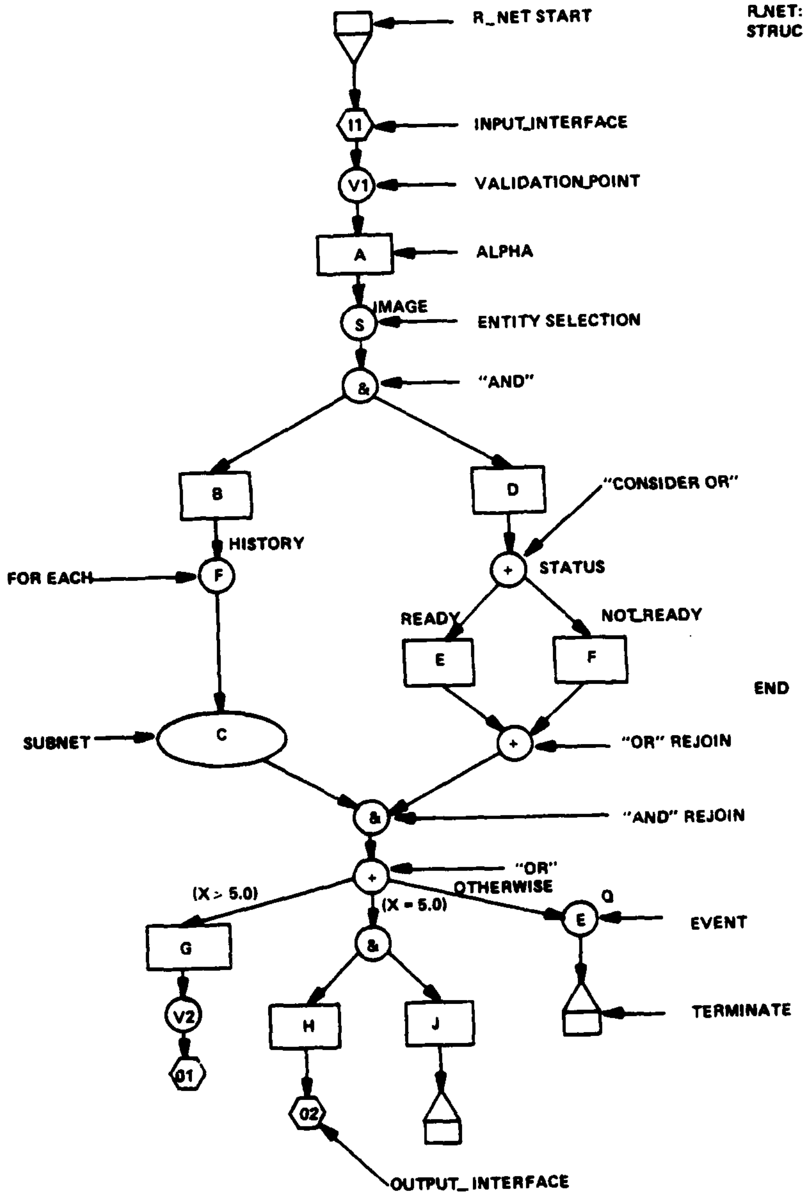
The language, RSL, is designed to allow a description of problem- rather than solution-oriented attributes and contains primitives for specifying structure in terms of processing flows, data, processing actions, and timing and accuracy requirements. Informative and descriptive material and management-related information may also be specified. RSL is an extensible language since certain primitive concepts are initially built in which can then be used to define additional complex language concepts. The primitives are elements, attributes, relationships, and structures. From these, a nucleus of concepts has been evolved through usage which, to date, has proven sufficient. The concepts of this baseline language consist of 21 element types, 21 attributes, 23 relationships, and 2 types of structures. Future users of the language can easily add to the nucleus by means of the extension features provided by REVS.

Analysis Tools

REVS is an integrated set of tools used to support the definition, analysis, simulation, and documentation of software requirements. A key concept of REVS is that all requirements are translated into a central data base called the Abstract System Semantic Model (ASSM). The RSL statements themselves are not stored in the ASSM. Instead, they are translated into representations of the information content of the requirements statements. This provides an efficient and flexible means of maintaining a large software specification in a relatively small computer data base.

The ASSM is a relational data base providing a common source for all requirements analysis, modeling, and documentation. The commonality of all data ensures that any combination of extractions from the ASSM at any time (e.g., a document and a simulation) will be mutually consistent. That consistency is essential to asserting that the requirements modeled in validation of the specification are equivalent in every sense to those written in the requirements.

REVS provides the mechanisms for entry of data into the ASSM as well as translation and interactive graphics, and a powerful set of tools for analysis termed Requirements Analysis and Data Extraction (RADX). Translation is the process of converting RSL statements into the ASSM information, where the



```
R_NET:  SAMPLE.
STRUCTURE:
        INPUT_INTERFACE   I1
        VALIDATION_POINT  V1
        ALPHA A
        SELECT ENTITY_CLASS IMAGE SUCH THAT (Y< Z)
        DO
                ALPHA B
                FOR EACH FILE HISTORY RECORD
                     DO SUBNET C END
        AND
                ALPHA D
                CONSIDER DATA STATUS
                IF (READY)
                     ALPHA E
                OR (NOT_READY)
                     ALPHA F
                END
        END
        IF (X>5.0)
                ALPHA G
                VALIDATION_POINT V2
                OUTPUT_INTERFACE 01
        OR (X = 5.0)
                DO
                     ALPHA H
                     OUTPUT_INTERFACE 02
                AND ALPHA J
                     ALPHA J
                     TERMINATE
        OTHERWISE
                EVENT Q
                TERMINATE
        END
END
```

Figure 2. R-Net Description

source of the statements may be cards, card images on tape, or keyboard entry from a terminal.

Interactive graphics are provided through a software package executing in conjunction with DataDisc Anagraph color graphics consoles to provide ASSM entry and documentation. It permits entry of structures and referenced elements in a manner parallel with the translator and may be used in conjunction with translation in an operational environment. Significantly, this allows the user to attribute graphical information to his structure, both for multicolor display on the Anagraph and for documentation via CalComp plots.

Information held in the ASSM may be selected for output using RADX. This tool is responsive to user direction in selecting either a re-creation of the information translated into the ASSM, or the formatted abstraction of that information in a user-defined HIERARCHY. The combination of these features allows complex selections to be effected, so that all information needed for documentation and that essential to configuration management can be abstracted from the system without the encumbrance of irrelevant data. Since all data abstractions are drawn from a common ASSM (and since that data base is confirmably consistent within itself), even redundant assertions in data extractions are absolutely consistent with one another.

Both static and dynamic analysis are provided by REVS in order to determine the internal consistency of the ASSM as well as its dynamic character. Static analysis is performed in RADX, which examines the data connectivity through the requirements to determine that the laws of logic and the conventions of the language are fully satisfied throughout. Some forms of completeness testing are also accomplished, determining, for example, that constants are provided as required; the scope of completeness testing is largely at the discretion of the user, since he may define extensive static analyses through RADX commands to supplement those inherent in the system.

Dynamic testing is accomplished by exercising the requirements against a model of the environment in which the system is to execute. Such simulations are provided by an automated simulation builder and a software package supporting its execution. Two different levels of simulation are supported: analytic, in which high-fidelity models of the environment and explicit performance measures are provided; and functional, in which the connectivity of the system is validated with nonanalytic models.

Status

The REVS system has been implemented on the Texas Instruments Advanced Scientific Computer both at the BMDATC Advanced Research Center (ARC) in Huntsville, Alabama, and at the Naval Research Laboratory in Washington, D.C. In addition, it is hosted on the CDC 7600 in Huntsville, Alabama, and at MDAC, Huntington Beach, California, and the CDC CYBER 74/174 TSS at TRW, Redondo Beach, California. The REVS software consists of 50K lines of PASCAL and FORTRAN code. With the overlay structure, REVS requires approximately 220K words to execute, including I/O buffers, stack, and heap space.

The methodology procedures have been documented, and research on this phase of the development has been completed. Experience is now being gained through the use of this methodology within the BMD community. The application of SREM to the distributed environment is being investigated.

Methodology Evaluation

The evaluation of any methodological approach for the development of software is an extremely difficult task within itself [5]. This difficulty is compounded as one considers measuring the effectivness of a phase of development such as requirements engineering, which has significant impact but relates only indirectly to the final product. Evaluation against a standard would be highly desirable but one does not exist. The "evaluation by doing" approach is usually limited by the dollars required to define, implement, test, and evaluate a system of sufficient complexity to demonstrate success. In addition, measurables during the experiment must be carefully chosen to allow effective evaluation and comparison. The double-blind or latin square approach to experimental evaluation helps very little in this area since the learning curve effect tends to warp results. Finally, since we are dealing with such a labor intensive activity, it is difficult to separate the inventive capability of the engineer from the effectiveness of his supporting tools. A good engineer supported by an excellent technology may exhibit the same character as an outstanding engineer supported by an average technology. With these views in mind, the SREM technology was evaluated with a light toward obtaining as much information about its character and capabilities as possible through a combination of evaluation and experimentation. The evaluation approach took the form shown in Figure 3 and consisted of an evaluation of effectiveness against known errors, evaluation of methodology capabilities, and an evaluation by doing. Effectiveness data have been generated for each of the areas [6]. Particularly noteworthy is the large reduction in requirements errors (Figure 4) being propogated to the next development phase.

The utility of SREM in requirements development has been demonstrated on a wide variety of projects [7,8]. These range from small problems designed to exercise one aspect of requirements development to large systems reflecting portions of defense systems. SREM has been also used in several roles for a wide variety of customers. Primary usage has been in a verification role, to verify a specification for the purpose of identifying problem areas, or in the role of defining new or expanded requirements capabilities. SREM was applied to the redefinition of requirements for the improved HAWK/TSQ-73 system [9]. This activity resulted in a complete test to the highly complex air defense arena. Assessment led to the identification of improvements that will increase SREM effectiveness.

The results of exposing SREM to divide user environment are summarized in the following paragraphs:

- Computer-aided requirements generation is extremely valuable in the reduction of commonly occurring requirements errors.

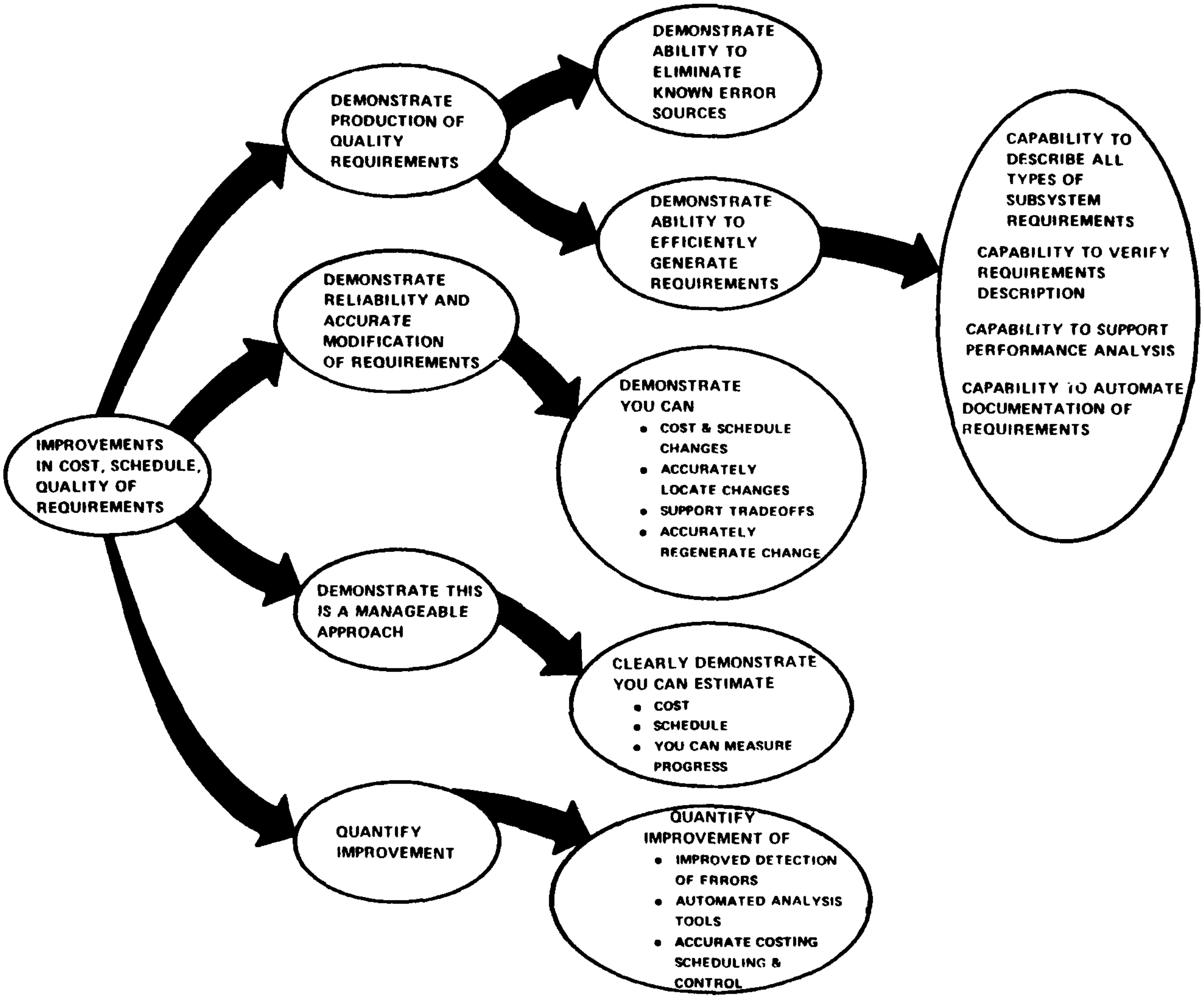

IMPROVEMENTS IN COST, SCHEDULE, QUALITY OF REQUIREMENTS
DEMONSTRATE PRODUCTION OF QUALITY REQUIREMENTS
DEMONSTRATE RELIABILITY AND ACCURATE MODIFICATION OF REQUIREMENTS
DEMONSTRATE THIS IS A MANAGEABLE APPROACH
QUANTIFY IMPROVEMENT
DEMONSTRATE ABILITY TO ELIMINATE KNOWN ERROR SOURCES
DEMONSTRATE ABILITY TO EFFICIENTLY GENERATE REQUIREMENTS
CAPABILITY TO DESCRIBE ALL TYPES OF SUBSYSTEM REQUIREMENTS
CAPABILITY TO VERIFY REQUIREMENTS DESCRIPTION
CAPABILITY TO SUPPORT PERFORMANCE ANALYSIS
CAPABILITY TO AUTOMATE DOCUMENTATION OF REQUIREMENTS
DEMONSTRATE YOU CAN
• COST & SCHEDULE CHANGES
• ACCURATELY LOCATE CHANGES
• SUPPORT TRADEOFFS
• ACCURATELY REGENERATE CHANGE
CLEARLY DEMONSTRATE YOU CAN ESTIMATE
• COST
• SCHEDULE
• YOU CAN MEASURE PROGRESS
QUANTIFY IMPROVEMENT OF
• IMPROVED DETECTION OF ERRORS
• AUTOMATED ANALYSIS TOOLS
• ACCURATE COSTING SCHEDULING & CONTROL

Figure 3. Proof of Principle

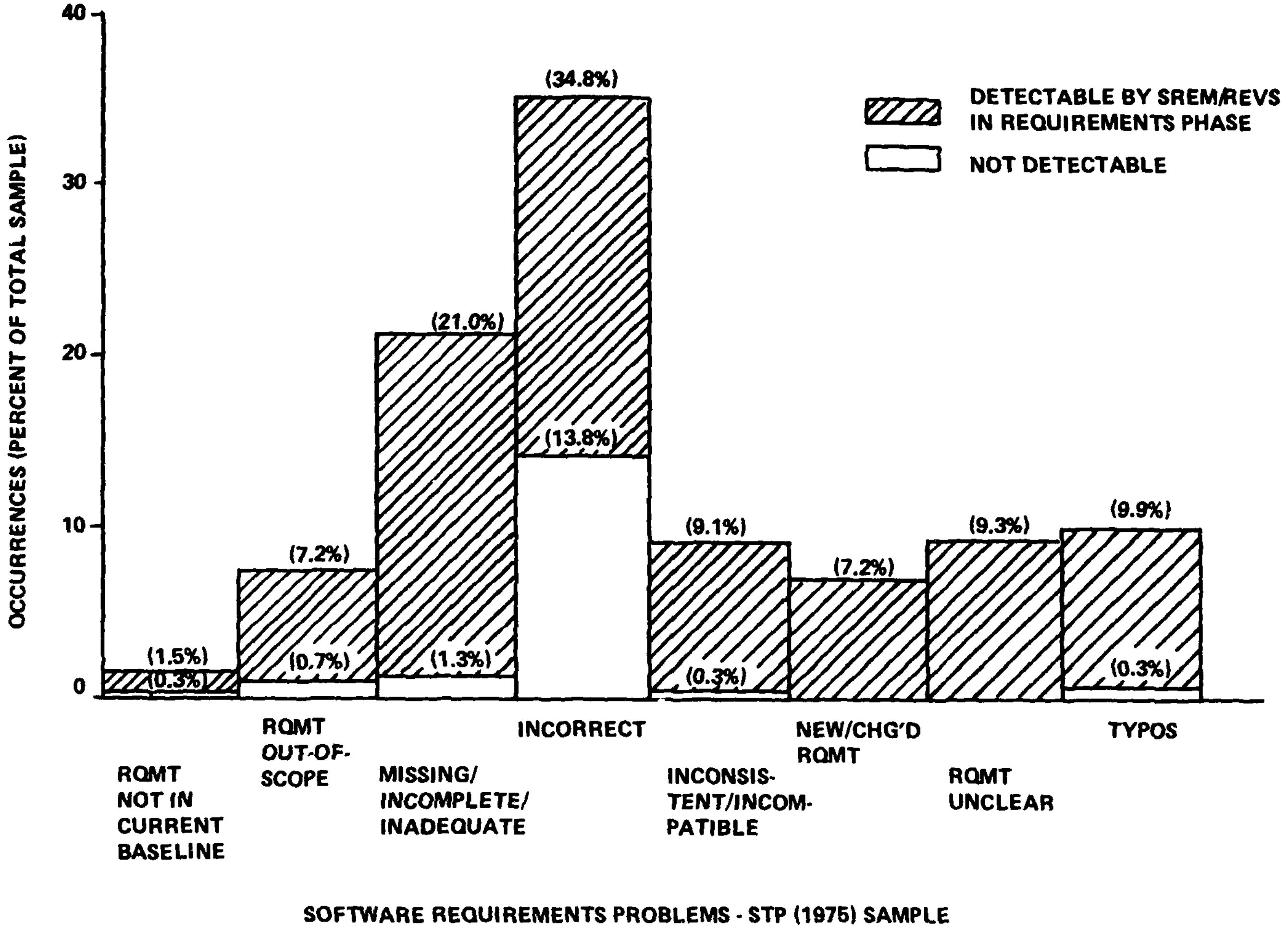


Figure 4. *SREM Error Reduction*

AD-A104 252 INTERNATIONAL BUSINESS SERVICES INC WASHINGTON DC F/G 9/2
US ARMY SOFTWARE SYMPOSIUM (2ND) HELD AT WILLIAMSBURG, VIRGINIA--ETC(U)
1978 S M TAYLOR DAAK70-78-D-0030

UNCLASSIFIED NL

2 OF 10
AD
A104252

- The computers required to host the support software are generally high-speed machines. The amount of checking and analysis required currently doesn't lend itself to small machine application.

- Even on large machines (e.g., CDC 7600 computers), a complex requirements statement may result in a data base that exceeds memory size. Simulations also soon exceed available storage. In addition, running times for complex searches through large data bases become costly in terms of complex time.

- SREM aids greatly in adding structure to requirements. This structure forces early consideration of the implication of data processing within an overall system context.

- The RSL has proven adequate to express requirements over a wide variety of projects. Extensions to the language identified have been within the capabilities of the extension features.

- Simulation for dynamic verification of requirements has proven effective, but additional capability and greater tie-in with the design phase would prove useful.

- Management and control of requirements development using SREM has been enhanced by the discipline of the computer as an in-line development tool. However, the improvements are a matter of personal taste and are not quantifiable.

SREM represents a giant step forward in the effective utilization of the computer as a development tool. Ongoing efforts are resulting in improved efficiency of operation and extensions have been identified which will allow its application to distributed data processing.

Acknowledgements

The thoughts presented on SREM have evolved over a number of years as part of the SDS development and reflect the views and contributions of a large number of people. Specifics on SREM are from documentation supplied by TRW Defense and Space Systems, Huntsville, Alabama.

References

1. C. G. Davis and C. R. Vick, "The Software Development System," IEEE Transactions on Software Engineering, January 1977, Vol. SE-3, No. 1.

2. M. W. Alford, "A Requirements Engineering Methodology for Real-Time Processing Requirements," IEEE Transactions on Software Engineering, January 1977, Vol. SE-3, No. 1.

3. T. E. Bell, D. C. Bixler, and M. E. Dyer, "An Extendable Approach to Computer-Aided Software Requirements Engineering," IEEE Transactions on Software Engineering, January 1977, Vol. SE-3, No. 1.

4. M. E. Dyer, et al., REVS Users Manual, SREP Final Report, Vol. II, TRW Defense and Space Systems Group, Huntsville, Alabama, August 1977.

5. C. G. Davis, "Requirements Problems in Large Real Time Systems Development," INFOTECH, State-of-the-Art Report, Structured Analysis and Design, 1978.

6. M. W. Alford, et al., SREM Experimental Results, TRW Defense and Space Systems Group, Huntsville, Alabama, 27332-6921-025, 1 June 1977.

7. M. W. Alford, "Software Requirements Engineering Methodology (SREM) at the Age of Two," Proceedings COMPSAC 78, Chicago, Illinois, November 1978.

8. R. C. Slegel, "Applying SREM to the Verification and Validation of an Existing Requirements Specification, COMPSAC 78, Chicago, Illinois, November 1978.

9. P. H. Browne, Jr., G. C. Hitt, and R. W. Smith, Utilization of SREM in IHAWK/TSQ-73 Requirements Development, TRW Defense and Space Systems Group, Huntsville, Alabama, 27332-6921-034, September 1978.

An Approach to Requirements Definition For Real-Time System

David Egli

U.S. Army Communications Research
& Development Command
Center for Tactical Computer Systems
Software Engineering Division

System requirements do not magically appear. They are derived from various requirement inputs, iteratively sorted, and hierarchically structured such that the system requirement definition can be understood.

The Tactical Executive provides the foundation from which a family of Secure Real-Time Tactical Operating Systems (TACEXEC) for computerized weapon systems could be realized. The TACEXEC provides an illustrative example of requirement definition methodology down to the modular level.

AN APPROACH TO REQUIREMENTS DEFINITION

FOR REAL TIME SYSTEMS

David Egli

US Army Communication Research and Development Command
Center for Tactical Computer Systems
Software Engineering Division
Fort Monmouth, New Jersey

I. Introduction

System Requirements do not magically appear. They are derived from various requirement inputs, iteratively sorted, and hierarchically structured such that the system requirements definition can best be understood.

The Tactical Executive (TACEXEC) provides an illustrative example of requirement definition methodology, down to the modular level.

II. Background

Traditionally, when the Army developed a new computer weapon system, the contractor was not required or encouraged to use any existing software previously developed. This led to the proliferation of many (170 systems) Ad-hoc, non-secure and unmanageable systems. This was the result of not having an all encompassing design philosophy which would provide the basis for a library of proven software programs, common across a spectrum of applications. The greatest cost saving could be realized in the area of Secure Realtime Tactical Operating Systems.

The Tactical Executive provides the foundation from which a family of secure Real-time Tactical Operating Systems could be built. The paper describes the requirement definition stage and a summary of resulting system design.

III. TACEXEC Requirement Definition

The TACEXEC requirements were derived (Figure 1) from the integration of three requirement sources: Tactical Mission Requirements, Tactical Application Requirements, and TACEXEC Project Requirements. The requirements were then ordered based upon what level of the weapon system hierarchy they best could be met (Section D). Summary of the input requirements are tabularized in Figure 2.

A. Tactical Mission Environment Requirements

Modern Army computer weapon systems are required to function under the most hostile conditions twenty-four hours a day. The system must be capable of defending itself against sophisticated attacks by a technically oriented enemy using probabilistic and crypto-analytic approaches, while defending itself from within against error prone application mission software, hardware failures, and "friendly troops" entering incomplete system commands and erroneous information. At the mission level, security threat monitoring, reliability, physical overrun, and processing speed is a major consideration in all tactical computer systems and thus was a major factor in the tactical mission requirements.

B. Application Software Requirement:

Tactical military systems for computerized communications, command-and-control guidance, surveillance, and target acquisition applications are generally characterized as consisting of a combination of critically time-constrained "real-time tasks which directly interact with the mission environment, "real-time" tasks which support the system operationally. and "non-real time" tasks which provide system diagnostics and maintenance capabilities. The typical tactical system includes provision for:

1. The acquisition of data from sensors, operators, external data links, etc.

2. The integration and compilation of this data into a data base to provide detailed information structures (tactical situation descriptive parameters).

3. The processing of these information structures for more precise quantitative descriptions of the tactical situation.

4. The distribution of data and/or control information for sensor and weapon, activation and operation. This includes the dissemination of information for display of the tactical situation for operators and users or transmission of data to remote systems.

5. System access control, authentication, and audit trails capability.

C. Tactical Executive Project Requirements:

The intent of the project was to develop a set of operating system primitives (Kernel) upon which a family of Tactical Operating Systems could be developed for supporting military application software. The TACEXEC was therefore required to be machine transportable, application independent, efficiently realizable using a Higher Order Language (HOL). A major concern was that the project requirement's would not impact the TACEXEC requirement goals. It later turned out to be the inverse case.

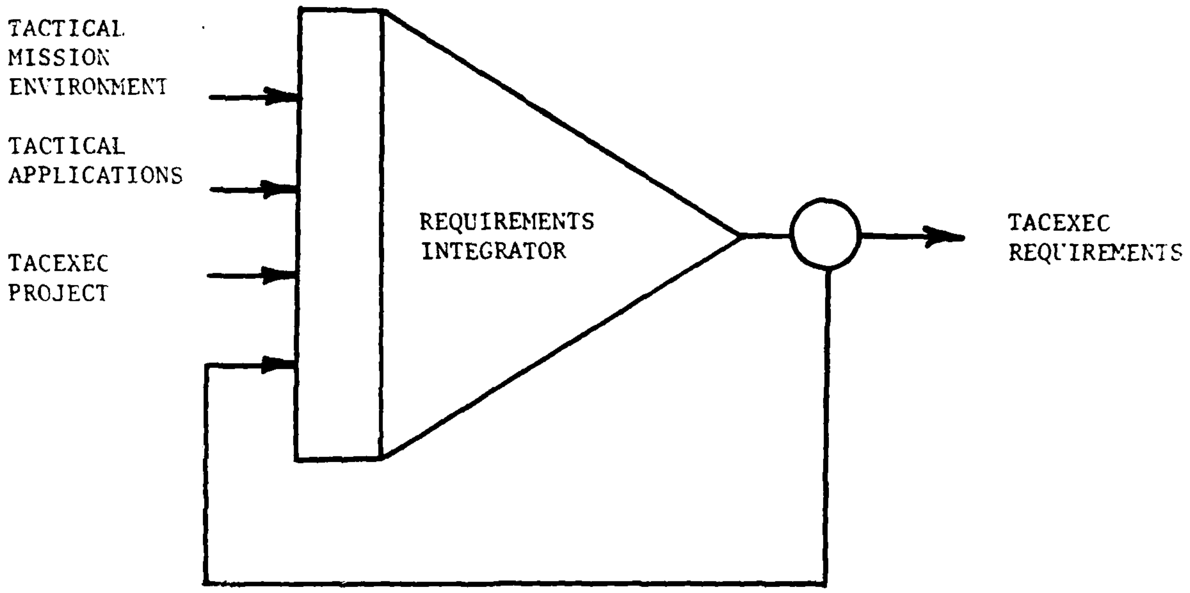


FIGURE 1 TACEXEC REQUIREMENT DEFINITION

Tactical mission environment requirements

- Reliability
- Security/Protection
- Capable of running in degraded mode
- High computational throughput
- Continuous operation

Application software requirement

- Real-time/near real-time tasking
- Access control
- Support multi-processor systems configuration
- Multi-level secure communications/ADP
- Reliability
- Drive non-standard military devices
- Protection/isolation of tasks
- Threat monitoring capability
- Exception handling
- Audit trails
- Support data Base Management Systems

Project Requirements

- Machine independence (CPU, I/O devices)
- Application independence
- Efficient realization of design
- Implemention using HOL

FIGURE 2 Summary of Input requirements

Tactical executive requirement definition

Real-time/non real-time tasking

Provably secure: Multi-level security model

Guarantee of service

Support family of real-time execution

High reliability

Support multi-processor configuration

Support process coordination

Support multi-processing

Support virtual memory

Support non-standard military I/O devices

Support Primitive File system

Machine independent

Application independence

FIGURE 3 Summary of TACEXEC requirements

D. Requirement Decomposition

The first pass is to isolate the common requirement set. These requirements will be met by the lowest abstract machine levels, (TACEXEC, Kernel). The second pass and subsequent passes isolate similar requirements into category sets (family operating systems). These form the middle abstract machine levels. The remaining form the upper abstract machine requirement set. Tactical systems do come in several flavors therefore requiring a family of operating systems. However, the family could be built from one Kernel (TACEXEC). It is therefore necessary to make one/several **last** requirement pass(es) defining the boundary between middle/Kernel abstract machine levels. The TACEXEC requirement (Figure 3) will be met by each of the five abstract machine levels described in section V.

Since requirements change in the real-world, only those abstract machine levels relating to the requirements changes will be affected.

IV. Security Requirement for TACEXEC

In the TACEXEC we wish to enforce a restriction on the way information may be passed from task to task. The particular restriction of interest is called Multilevel security. Each process has associated with it a CLEARANCE and a CATEGORY SET. The system has a fixed finite number of clearances that are totally ordered by the relation "less then." For example, the clearance CONFIDENTIAL is less than SECRET, which is less than TOP SECRET. For convenience, clearances are represented as integers.

The definition of the multilevel security model can be stated simply. For any f_1 and f_2 in F: where f_1, f_2 are functional references in domain F, K and I are the security and integrity levels respectively.

$$f_1 \rightarrow f_2 \quad \Rightarrow \quad K(f_1) <= K(F_2) \quad \text{AND} \quad I\ (F_1) \quad >= \quad I\ (f_2)$$

This simply states that if there is any possibility of information transmission between two function references, then the transmitting function reference must have a security level less than or equal to that of receiving function reference, and the receiving function reference must have an integrity level less than or equal to that of the transmitting function reference.

In other words, information can only flow upward in security or remain at the same level, or only flow downwared in integrity or remain at the same level.

The formal specifications for the TACEXEC were written in SRI International specification language SPECIAL. The security model was confined to two modules READ-ALLOWED, WRITE-ALLOWED and enforced in following five TACEXEC modules.

virtual memory
file system
user/system I/O
dispatcher
Synchronization primitives

An example of the formal specification of secure requirements module READ-ALLOWED and WRITE-ALLOWED is given in Figure 4. The following SPECIAL nominclature is as follows:

=	equal
<=	less than or equal
>=	greater than or equal
|	such that
=>	implies
$	comments

The definitions consist of: abstract data type and Boolean Algebra and is beyond the scope of this paper.

```
TYPES

clearance:  (INTEGER i | 0 < i and i  <= max_clearance );
category set:
( VECTOR_OF BOOLEAN cs  |  LENGTH (cs)  =  number_of_categories );
access_level:
STRUCT_OF(clearance security_clearance;
          category_set security_categories;
          clearance integrity_clearance;
          category_set integrity_categories);

PARAMETERS

INTEGER max-clearance $( the highest clearance),
number_of_categories,

DEFINITIONS

BOOLEAN read allowed(access_level subject_al, object_al)
IS subject_al.security_clearance
 >= object_al.security_clearance
AND subject_al.integrity_clearance
 <= object_al.integrity_clearance
AND(FORALL INTEGER I | 0  < i AND i   <= number_of_catergories:
( object_al.security_categories [i] )
=> (subject_al.security_categories  [i])
AND (subject_al.integrity_categories  [i] )
   =>object_al.integrity_categories  [i] ));


BOOLEAN  WRITE-ALLOWED (access-level subject_al, abject_al)
         Is READ-ALLOWED (object_al, subject_al);
```

FIGURE 4

V. TACEXEC DESIGN REQUIREMENTS

A. Real Time Behavior

The intent of TACEXEC is to be responsive to the needs of tasks. For real time tasks this need is related to tasks being served within a certian time frame. Among the tasks that might be served by TACEXEC is a scanning radar, delivering signals at regular intervals. Another task could be a fire control system, that requires extensive service only in bursts. Message transmission is another task, that is typically of low criticality except that there might be a maximum delay that is acceptable for the transmission of a message. Each of these tasks poses different needs on TACEXEC leading us to three classes of tasks: interative, demand, and background.

How does the system guarantee service requirements, particularly for the interactive and demand tasks? That is, how is it assured that the system loading is sufficiently low such that the service needs will be met, but not too low so as to preclude the inclusion of additional tasks that could be handled. A scheduling algorithm was developed that can accommodate maximum loading of the system. However, the processing time required for this scheduling algorithm most likely precludes its use in any real time system. On the other hand, if a scheduling algorithm based on tasks priority -- an easily implemented algorithm -- is used then the system can be loaded such that in excess of 60% of the time slots will be guaranteed to be available for task processing.

B. Functional Capability

In general, the interface (Figure 5) provided by a real time operating system need not be as powerful as that for a general purpose time-sharing system (e.g., Multics). However, a real time operating system is intended to execute collections of interacting programs and should have sufficient functionality to realize some reasonably complicated subsystems. The TACEXEC provides the following features at the user interface.

- virtual memory consisting of address spaces and segments
- a file system
- processes
- synchronization primitives

PROCESS COORDINATION

P
V

USER I/O

Set Event
Read Device
Write Device
Send Command
Receive Status
Change Status

PROCESS PRIMITIVES

Create - Interative-Process
Create - Demand - Process
Create - Background Process
Create - Event
Delete Process
Occurrence
Block

FILE SYSTEM

Create File
Delete File
Load File
Unload File
Append File

VIRTUAL MEMORY

Create-address-space
Delete-address-space
Create-segment
Delete-segment
Segment-read
Segment-write
Get-segment

SYSTEM I/O

Set event
Read-device
Write-device
Receive status
Device input
Device output
Device command
Change status

DISPATACHER

Create-process-identifier
Create-event
Create-wakeup
Block
Schedule interative process
Schedule demand process
Schedule background process
Stop process
Continue
Wait

FIGURE 5 TACEXEC Visible User Interface Functions

Conspicuously absent from the system are: directories, linkage sections, support for procedures, among other things that are in Multics. It should be noted that these later facilities would be built out of the TACEXEC facilities providing the bases for a family of realtime operating systems. (Thus, TACEXEC can be viewed as a kernel out of which a family of operating systems could be constructed).

C. Efficiency

As indicated above, a classic principle underlying an operating system is the efficient management of resources (cpu, disk, main memory, etc.). In a real time operating this principle is in conflict with, and of secondary importance to the guaranteeing of service to tasks. In particular, the efficient management of tasks often introduces nondeterminism such that accurate performance prediction is not possible. Fortunately, the critical tasks (iterative and demand) typically require little memory and cpu time for each execution. Also, there is little sharing of I/O devices in a real time environment. Thus, such tasks can be given total access to all needed resources of the system for the short time required for execution. Also, if the program and data for these tasks is retained in main memory, then it is possible to guarantee (by proof) that the service needs of these tasks are met. This characteristic of the tasks led us to the decision that the virtual memory system is to be totally resident in main memory.

There are other issues regarding the efficient realization of TACEXEC. A high level language is desirable for casing the burden of implementation and to aid in portability. However, there are important features of a high level language that relate to the efficient implementation requirements.

In addition, there are efficient hardware support for certain functions is required for a real time system, e.g., context switching.

D. Security

TACEXEC is intended for an environment where multiple users have simultaneous access to the system, and wish to be assured that their information is not available to certain other users -- on an accidental or intentional basis. That is, the system is not to be a vehicle for the erroneous handling of information. For a military application, the *multilevel security model* seems appropriate (Section IV). In this model, each user has a *clearance* and a *category set*; the cartesian product of clearances and category sets define a partial ordering of *security levels*. The values for clearance are the conventional classifications: UNCLASSIFIED, CONFIDENTAIL, etc. The categories represent an orthogonal restriction, and include such "controls" as NATO, ATOMIC. The model requires that information stays at the same security level or flows upward.

The model also includes integrity which provides additional restrictions on the flow of information. For example, based on the security restriction alone, there are no limitations to the "upward" transmission of information. That is, the model does not prevent the "destruction" of a SECRET document by an UNCLASSIFIED user. The inclusion of integrity places limitations on such modification.

E. Provability

TACEXEC has been designed to be provable, in partucular by a mechanical (computer program) program verifier. The main properties of concern here relate to security and guaranteeing that tasks will receive the promised service. Other properties, also of interest, relate to guaranteeing that the user interface operations provide the intended functional behavior.

The development of system according to SRI International's Hierarchial Development Methodology (HDM) is accomplished in stages. For example, in the specification stage, each of the system modules (a module is provided for each "facility") is formally described by a specification. In the implementation stage, the operations of each module are implemented by a program. A proof is associated with each stage. For example, it is possible to prove that the multi-level security model is satisfied by the specifications of the modules of the user interface of TACEXEC. It is also possible to prove that the guarantee of service property is satisfied by the user interface specifications. Separate proofs can demonstrate that the programs are correct with respect to the specifications.

This separation of proofs serves to simplify the overall proof process, as any useful decomposition of effort should do. In addition, it limits the amount of reproving that must be done as the system evolves. For example, a change to the implementation (possible to install TACEXEC on a different processor) does not require any change to the specification proof if the specifications are left intact.

F. Portability

No real time operating system can be totally portable. In order to achieve efficiency, there will alway be machine-specific code. The Army's concern was to produce a system where the amount of effort required to move TACEXEC from a machine on which it is successfully executing to another machine is small.

Much of the effort involved in developing a system is associated with "design". In general, design is concerned with deciding what a system is to do, avoiding details on how it is done. In HDM, the initial stages are concerned with design, while the later with realization. The output of these early stages is a set of specifications for the modules and a precise description of the structure of the system. These can serve as the design for TACEXEC independent of the hardware on which it executes.

Furthermore, a system developed according to HDM is usually designed as a hierarchy. (The TACEXEC hierarchy consist of five levels). Typically, the modules of the upper levels are implemented by software, the middle levels by a mixture of hardware and software, and the lower levels by hardware. Thus, even the implementation affords a measure of portability, in that many of the programs will remain intact in going from one hardware to another.

Nuts & Bolts of Software Acquisition

Thomas A. Rorro

U.S. Army Electronics Research & Development Command
Beta Joint Project Office
Integration Division

The Government is a major consumer of computer software. The trend in system development is to imbed computer resources in all tactical systems. The manager of these systems must understand the limitations of computer software in order to effectively manage its development.

The expanded flexibility and capabilities of software make it desirable for use in tactical systems. At the same time, however, these attributes amplify human inefficiencies and create potentially grave problems in systems development.

The "Nuts & Bolts of Software Acquisition" address the cause of software development problems and provides a "cookbook" approach of their resolution. Top down design methodology is applied to system development. The elements of a software development contract are described in detail. In addition, the critical aspect of schedule which enforces the top down approach is described.

NUTS AND BOLTS OF SOFTWARE ACQUISITION

Thomas A. Rorro

Introduction

Congratulations, you've been selected to develop a new system for the US Army. The first questions to be asked are: What is it?, and How does it fit into tactical scheme of things? Unfortunately, the more people you talk to the more diverse the answers become. The only thing that's for sure is that the system contains a computer and it will be better than what the troops have now!

There is one more thing for sure. "You are headed for big trouble if you're not careful!" There is hope however; and the intent of this paper is to demonstrate that your problem can be solved by simply applying common sense. In fact, software acquisition is as simple as A, B and C.

1. The Roots of All Evil

A. Flexibility.

Before the advent of computer technology, all system functions were implemented in hardware. This technology severely limited the complexity of the system functions which could be performed. Simplicity permitted the design to proceed without the need for rigorous controls. Hardware development by this method is efficient. The problem comes when a less than rigorous controls are applied to software development.

The computer and its software provide the system designer with a new flexibility. Changes can be made with the stroke of a pen. But, the error in a single instruction cause the system to fail. The problem does not lie with the computers interent flexibility. It lies in the designers inability to completely describe his design and to communicate with the user. The computer's flexibility simply amplifies this defect to the point of severe consequence. The situation can be likened to the building of the biblical tower of Bable.

B. Doctrine

There are two key elements in any system. The first is the mathematical functions which it performs. And second, is the doctrine which defines its use.

In the past, a system was developed in response to a users need. The developer applied the available technology and produced a system which implemented the mechematical functions required. The system was then presented to the user who created the doctrine. These processes were not completely independent but the interplay between the user and developer were minimal.

The advent of computer technology completely changed this relationship. The computer is capable of implementing both mathematics and doctrine. The two step process of mathematical system development followed by doctrine development is now a single process. Because of this, the interplay required between the user and developer has grown exponentially. And with this growth comes the risk of fielding a system which is neither mathematically sound nor meets the users needs.

C. Creativity

The flexibility and doctrine capabilities of the computer provides the system architect a new opportunity to be creative. In the past, he could be alone in his portion of the design and exercise creativity to the limits of the technology. Now, the architect must share the system development with the user. They must work together and be creative in their own area of expertise. There is, therefore, a firm requirement for a formal design mechanism.

In the alternative, as men learn the system problems they begin to exercise creativity. Often, this begins before they have completed the learning process. As time passes, the developers become creative users and the users become creative developers. Before long and with some luck, they will each have reinvented the wheel.

D. Maitenance

Perhaps the last, but by far the most important phase of the system life cycle is deployment. Once the system is in the hands of the troops, they are sure to find hundreds of previously unimagined ways to make it fail. It is the responsibility of system developers to provide support documentation for the deployment phase which will provide for system maintenance.

The flexibility provided by the computer results in the requirement for detailed documentation. At some point in the development cycle it will have to be procured. The question is; When?

2. Most for the Money

In order to have a successful system, the developer must properly deal with the issues of flexibility, doctrine, creativity and maintenance. Timely documentation can provide the necessary elements to resolve each of these issues.

The documentation represents a complete description of the system. As such, it can be of great value in coordinating the efforts of the developer with the desires of the user. It can provide a formal mechanism for the design and the visibility to implement rigorous design controls. The technique of applying documentation to this end is called top down design. A significant factor of this approach is the cost which is born early in the development cycle. Essentially, the cost of documentation is the same regardless of when it is performed. The benefit of top down design is its ability to deal with the issues of flexibility, doctrine and creativity at little additional cost. As such, it represents the most for the money.

3. Top Down Design

In general terms, top down design is simply the development of a system starting with the broadest definitions and working to the minute details. It is necessary to partition this effort so that measurable milestones are available. Taking industry, developer and current procedure into account; a three tier system has proved best.

The first tier is the functional description (A level specification). This document is produced by high level personnel with significant user interaction. The final document should be meaningful to everyone affiliated with the development.

The second level of development is the Input/Output and mathematical description (B level specification). The user is concerned with the Input/Output and in particular with the man-machine interface. The engineers and scientists are concerned with the mathematics as well as the cohesive aspects of the system.

The third tier of development is the implementation (C level specification). This document is of prime concern to the programmers with only minor involvement of the engineers and scientists. The C specification contains a description of the system in detail including the listing of the software program.

4. The Carrot and the Stick

There is the age old problem of getting what you want when you want it. In order to insure that system development is proceeding smoothly, the developer needs both visibility and control.

Visibility is achieved through the documentation. As each level is completed it is reviewed by all parties. Formal concurrence represents the completion of that major milestone. It is appropriate to tie payments and awards to successful completion of each level of specification. Thus the incentive of on-time performance and a quality product is provided.

A system is not built until the software is programmed. But effective programming cannot begin before the B level specification is complete. Control can be achieved by restricting the programming through a contractual stop. A statement to the effect that "programming shall not commense in any form prior to receipt of an authority to proceed release from the Government" will provide the required control. This release should be granted incrementally as each draft C level specification is produced. The control aspect of software development is critical to its success. It is imperative that the authority to proceed release be given only after the design is complete. To allow programming prior to this point is to invite disaster.

5. Contractual Elements

The implementation of top down design and the required controls involves several elements. These elements include the specifications, design reviews, testing, and a software warranty. Each of these elements is explained in detail in the paragraphs which follow.

A. The A Level Specification

Software design and development must start from the top. Generation of the A level specification begins shortly after contract award. It is the first in the series of software design documents which describe the system in progressively greater detail. The A level specification breaks the system into its main functional areas. The requirements of the Request for Proposal (RFP) are reflected in this document. The A level specification provides a complete system description including both accuracy and functional requirements. Written properly, the A level specification could replace the RFP and any other pertinent government documents. The document is designed to be relevant and completely understandable to the highest level user and developer personnel.

The A level specification represents the contractor's understanding of the system and its requirements. The delivery of the A level specification represents a critical milestone in the development. As such, a design review is required.

B. Functional Design Review

The Functional Design Review (FDR) is the mechanism by which the government and contractor reach agreement on the requirements and capabilities of the system. The results of the review are incorporated into the A level specification and the document is formally approved by the government. At this time the A level specification is subject to formal configuration management controls. The A level specification becomes the baseline for development of the B level specification.

C. B Level Specification

Now begins the detailed design of the system. The function breakdown of the A level specification is further divided into the modules of the B level specification. The function and capabilities of each module is specified with appropriate references to the requirements of the A level specification. The main function of the B level specification is to provide the complete mathematical description of the system. There is no separate design process. The development of the B level specification is the system design process and the B level specification is the documentation of the design.

The B level specification is an important and detailed document. It is made up of two volumes. Volume I consists of four parts: the User's Manual; the Operator's Manual; the System Outline; and the Preliminary Data Base Specification. The System Outline section is a system overview description of the entire software system without delineating to specifics of each independent module. The User's and Operator's manuals provide a concrete description of the man-machine interface at a time convenient to provide orderly input to the system implementation (C level design phase). The Preliminary Data Base Specification raises the issue of orderly data management. It is a preliminary specification because all the problems will not surface until the C level design begins. However, since orderly information flow is essential to an efficient software design the Preliminary Data Base Specification is a valuable asset to the B level system design. Volume I of the B level specification is of interest to both the user and developer and should be carefully reviewed by both.

Volume II of the B level specification is the detailed description of the system and its modules. It consists of the specifics of each program and is prepared in accordance with Appendix VI of MIL-STD-490 supplemented by Appendix VI of MIL-STD-483.

Volume II of the B level specification is the critical document for review by the developers software and system specialists. All the major design decisions have been made and should be evident in this document. Its completion is a major milestone in the software development effort. A preliminary design review is held to resolve any discrepancies between the B level specification and the A level requirements.

D. Preliminary Design Review

The function of the Preliminary Design Review is to provide a mechanism for the government and contractor to reach agreement on the final system configuration. The results of the review are incorporated into the B level specification. The B specification is then submitted for government approval. Formal configuration control of the B level specification can begin at this time. There is an option however, to allow the contractor additional latitude by requiring only limited configuration management. Under this approach changes can be made by the contractor to the B specification. He is required to inform the government of all changes. Since government approval for all changes is not required, the cost implications can be minimized. This technique must be carefully considered in the light of system development status and the risk factor must be carefully weighed. The completion and approval of the B specification sets the stage for detailed design and coding to begin.

E. The C Level Specification

The modules defined in the B level specifications are further decomposed and arranged to provide for efficient mechanization in the computer. Numerical techniques are used and trade-off analysis is performed to implement efficiently the requirements of theB level specification. The result of this analysis is a partial C level specification. A Critical Design Review is held for each partial C level specification. This review allows the government to insure compliance of the C level implementation with the A and B level requirements.

Approval of the partial C level specification represents the authority for the contractor to start coding. The listing of the code, the test results together with the partial C level specification combine to create the complete C level specification.

F. Critical Design Review (CDR)

The CDR allows the government a final point for review and control of the design before it becomes a software program. All the fine details of the module under review are available. The problem areas should be visible to government software system analysts. Close coordination between the government and the contractor should provide for cost effective solutions to the problems which are uncovered.

G. Preliminary Qualification Tests

The completion of the C level design and implementation of each module will proceed at different rates. Since the modules are defined to be independent a method of incremental testing is appropriate. The PQT is the test performed on each module. The order of testing of the modules is defined in the PQT plan. The optimum PQT method is to build up the system by testing the new module with all previously tested modules. This method provides controlled system integration and testing. When all modules have been tested and integrated the Final Qualification test is performed.

H. Final Qualification Tests (FQT)

The FQT is the system test. This test is performed in a two stage process. Due to the expense of field testing a simulation test environment is recommended for the initial stage. Upon successful completion of simulation test the contractor is permitted to begin the second stage; field testing. The results of both simulator and field testing are then documented and the system is delivered to the government.

I. Physical Configuration Audit (PCA)

The government inventories and accepts all contract deliverables at the PCA. There is no reasonable method to completely and accurately audit the software. The government is assured of its quality by his previous careful review and participation in the design process. The PCA should be only a formality which consummates the government's efforts during the term of the contract.

J. Software Warranty

Since validation is not practical, the government should require the contractor to warrant the software. The software warranty will require the contractor to fix and document software problems which occur during a period of time after government acceptance. The warranty period may also be of value as a means of educating an independent government agency in the system's software maintenance procedures. This method provides the government with insurance against problems which are not uncovered in the sterile test environment.

K. Validate and Verify (V&V)

It is important to provide check and balances on any software development. V&V represents a method to insure the quality of the software product and the documentation. The cost of V&V can be astronomical if carried to extreme. Experience indicates that the cost of each uncovered error grows exponentially. A practical approach to V&V is to set aside a small sum of money for an independent contractor. If his efforts are fruitful continue the process. When the cost of his effort exceeds the useful result the V&V effort should be terminated.

L. Schedule

The software development process of the A, B and C level specification is serial. Within each level, however, parallel development of the individual modules can be performed. This allows the contractor to apply manpower in an efficient manner but preserves focal points in the design.

In the ideal case, the contractor effort should be suspended after each submission until government review and approval is accomplished. This provides the government with positive control of the software development.

If the contractor is allowed to begin the next phase of development before the previous phase is approved the monitoring process becomes more tedious. Basically, the government makes the assumption that there will be no major problems with the contractor's software development. This is a function of the software abilities of the contractor, and a careful analysis of the risk is recommended before selecting this approach.

An example of a software development schedule is displayed in Figure 1. The time allotted for each development will depend on the nature of the system and the technical information furnished by the government.

6. Summary

There is an old adage "You get what you pay for." It is as true now as ever and highly applicable to software acquisition. There are no magical short cuts. The best way to approach the problem is through an orderly, step by step procedure. By this approach, a development can run for a year or longer without demonstratable hardware. It is a tough course to follow with all the pressures to get the job done. Now, stick to your guns. The top down approach is the only low cost, low risk method. It represents the difference between success and failure.

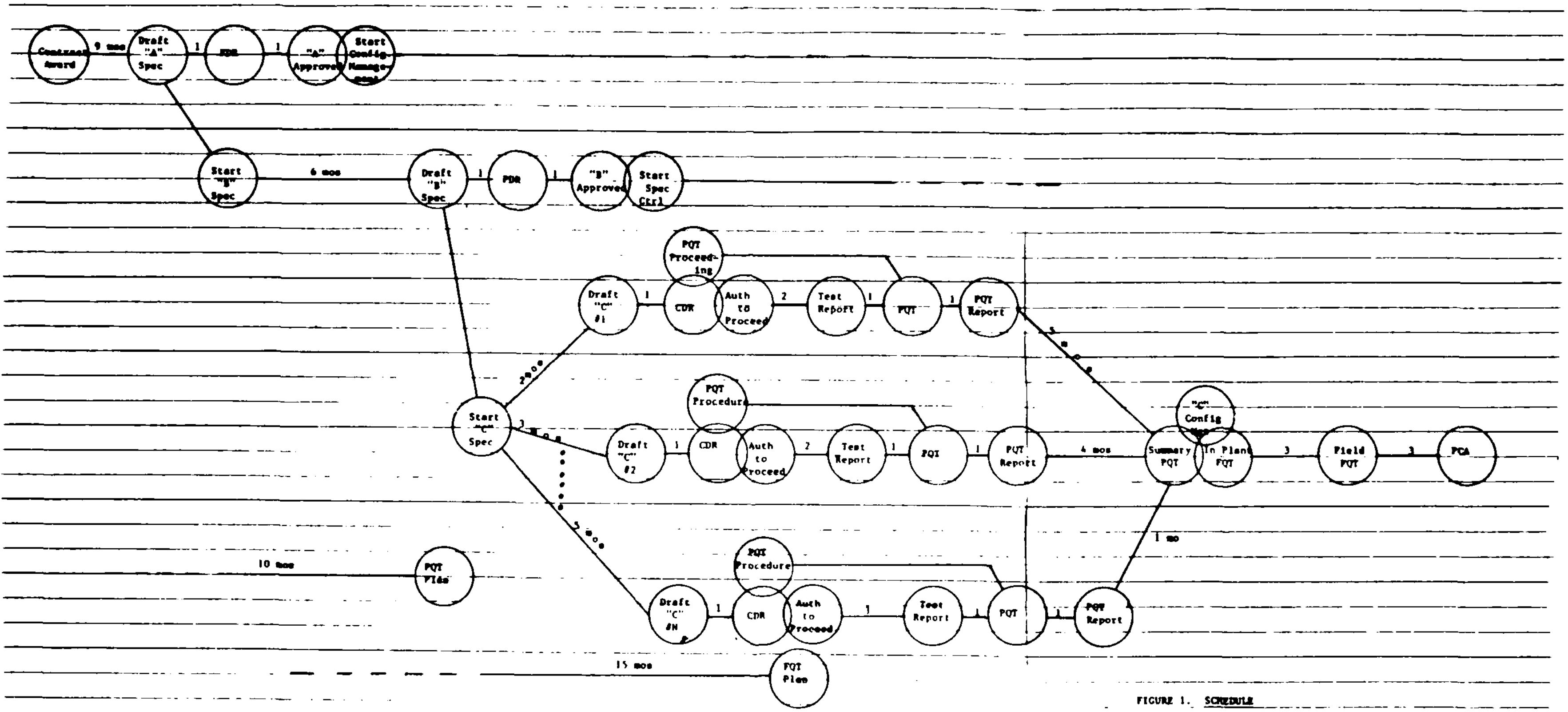


FIGURE 1. SCHEDULE

LIFE CYCLE MANAGEMENT

MAJ Edward H. Ely
AIRMICS

LIFE CYCLE MANAGEMENT

SESSION CHAIRPERSON: Edward H. Ely

AIRMICS

SESSION SUMMARY

Preparation for future software life cycle management planning requires serious contemplation of those factors which comprise the entire life cycle process. Dr. Victor Basili (University of Maryland) began the session by reviewing several life cycle elements in an attempt to provide a better understanding of such life cycle factors. Included were reviews of the roots of data collection; an examination of the meaning of milestones; discussion of life cycle dynamics and metrics; consideration of the impact of tool usage; and the problems of technology transfer. Dr. Thomas G. DeLutis (NRC) continued the session with a survey of software life cycle complexity and the important role of modeling in early determination of such complexity. The ability to provide meaningful estimates relative to various types of problems, plus the iterative nature of the life cycle, were also discussed. Dr. J. David Naumann (University of Minnesota) concluded the session with a challenge to the "single-life-cycle" concept commonly used to determine information requirements. Contingencies which define the uncertainties in the determination of information requirements, and alternative strategies for information requirements determinations at different levels of uncertainty, were also described. The methods put forth were no cycle, linear life cycle, recursive life cycle, and prototype.

Toward Understanding the Software Life Cycle

Victor R. Basili

Department of Computer Science
University of Maryland

There is need for a better understanding of the software life cycle process with respect to its management parameters and the resulting product. This understanding assumes some basic theories about the activities of the life cycle process and the interaction of these activities, a set of standard definitions for each of the activities and terms involved in the process and resulting product, a taxonomy for each of these terms which parameterize them for a particular environment or point of interest, and the collection of data exposes what is actually occurring with respect to each of these taxonomies for each of the terms. The development of models and measures are then possible based on analytical theories or empirical studies. These models and measures need to be validated for a variety of projects based upon each of the taxonomies using data collected during the life cycle process for each project. This should result in the refinement of the theories for the local environments and the development of new models and metrics.

Once a basic model of the life cycle process has been developed with its associated management and development methodologies, it must be transferred into practice by building tools for the manager and programmer based upon these associated models. Data must be collected to give to the manager information on the progress and quality of the product. These methodologies can then be transferred from project to project, using the developing data base of quantitative knowledge about the software life cycle as background for management decisions on the current project and to help further refine our models and metrics for future projects.

This paper discussed the basic components necessary for understanding the software life cycle, the model of the dynamic nature of the development process, the requirements for transferring technology from theory to practice and from project to project, and some areas where models and metrics currently exist.

TOWARD UNDERSTANDING THE SOFTWARE LIFE CYCLE

Victor R. Basili
University of Maryland

Introduction

There is need for better control in the management of software. This control can be achieved only by a better understanding of the software life cycle process in theory and a more effective method of exposing progress in practice. Understanding assumes models and measures of the various aspects of the process and the product which parameterize each of the factors that influences them. It requires identifying these factors, knowing their bounds, and measuring their interaction. Visibility assumes the ability to produce certifiable checkpoints of progress. In many cases, this requires the availability of data, collected during the development process, that conforms to the given models and measures.

Current problems arise from the fact that there are too many parameters affecting the development of software. It is difficult to isolate all these parameters, and understand their effect and the relationships and interactions they foster in the development process. Even those factors that are intuitively recognizable cannot be defined or quantified very well. It is almost impossible to find an agreed upon set of definitions for some of the most basic factors or terms; e.g., size, specification, etc. We do not have a realistic model of the life cycle process itself. Most views of the life cycle are in terms of static phases; e.g., requirements, specification, design, coding, testing and maintenance. In practice, the life cycle process is not static butdynamic, iterating through all phases. The phases define aspects of the development process, but the sequence of phases does not describe their interaction. When we try to fit the process to this static model, we are left with a poor understanding of the dynamics of the process.

Visibility of progress is hard to assess, especially during the early stages of development. Milestone definition is vague and it is difficult to verify that the product at any point in time corresponds to what is expected. This is partly because progress doesn't correspond to the static model and there is not enough solid information to permit confident assessment of success.

To solve the problems of control, understanding and visibility, we must develop a science of the software life cycle process. We must develop realistic models that fit the process. We must develop quantifiable measures that accurately feed back information on progress and quality. To this end, we must collect data for the development and validation of models and measures for feedback and visibility of the process and product.

In what follows, some of the basic problems with model development and data collection will be discussed in more detail. An approach for building and using realistic models will be given along with its benefits in the development process. Finally, some promising models and measures currently in existence will be given.

Problems

The major problem in model development is that there are too many parameters and factors to consider. There are (1) people-related factors, e.g., the number of people involved, the level of expertise, previous experience with the problem, group organization; (2) problem-related factors, e.g., type of problem (data base manipulation, real time, etc.), proximity to the state of the art requirements, susceptability to change, number of external interfaces; (3) process-related factors, e.g., top down design, use of librarian, programming languages, reporting mechanisms; (4) product-related factors, e.g., deliverables, real time requirements, number of modules, size, efficiency tests; and (5) resource-related factors, e.g., budget, deadlines, response times, target and development machine systems. We need to understand the effect and boundaries of each of these factors in theory and then map them onto the current problem to be solved in practice. We need to know how to balance various factors and make adjustments.

Understanding these factors, their boundaries and interactions requires analyzing their effect on a large number of projects. This requires the collection of the appropriate data and the ability to transfer this data from project to project across government agencies and industry, in order to build enough of a data base to isolate common developments and compare the results. However, there are major problems in data collection and transfer of information. First we must know what data to collect before we begin collecting it. It must be model driven. (Data collection can be very expensive if it is not done right. It need not be if it is well planned as to what is wanted and how it is to be processed.) Second, basic to data collection and information transfer is a standard definition of terms and taxonomies. Neither of these currently exist. In fact, we do not agree on such simple definitions as what constitutes source code, project size and error. It is not even agreed on what is meant by the life cycle terms, that is, the definitions of the terms given in the introduction are not standard. When reading the analysis of a particular project in the literature, one cannot always transfer that knowledge to another project because it is not always clear what the original author meant by many of the terms which were involved and what their values were. Many of the contributing factors are not explicitly given.

Besides the definition of basic terms, we need taxonomies; that is, we need to understand the data at various levels of abstraction for different classification schemes and different environments. At levels of abstraction, we need to know when to worry about details and when to see the big picture; e.g., budget factors can be viewed for the entire project across the entire life cycle, or for some subcycle (such as the design phase or the design, coding and testing phases), or for some subsystem (such as the design phase of a particular component).

We can categorize a product or subsystem by its type (e.g., data manipulation, mathematical, real time). This permits the specific weighting factors for productivity which can be associated with the type of application. We can classify an environment by its constraints (e.g., execution time critical, storage critical, calendar time critical). Then we can better understand the effects on budget or development time based upon the specific set of constraints on the problem.

We are usually interested in collecting data for a specific purpose. How we classify data dictates the kind of data collected. For example, there are many ways to classify error data: by cause (e.g., misunderstanding requirements, misstated specification, clerical coding error), by time used to fix, by technique used to find or correct the error, etc. Each of these classifications requires the collection of different data and tells us different things about the development process itself. In evaluating a technique such as code reading, we would like to know what class of error it minimized or eliminated. We would like to know what class of error is most common to a particular type of software product or particular methodology.

Until we can agree on a standard set of definitions of terms and a standard set of taxonomies for categorizing data, it will be difficult to understand and analyze the software development process, to develop models that can be parameterized for different environments, and to gain real control of the management process.

Part of our lack of understanding also stems from a poor model of the life cycle process itself. Milestones are often associated with the completion of a static set of phases. Typically, these phases are (1) requirements--an operational description of the user's needs; (2) specification--the developer's abstraction of the requirements giving what is to be done without indicating how; (3) design--the abstract description of how a system is to work; (4) coding--the implementation of that design in machine executable form; (5) testing--the verification that the operating coded system satisfies the operational requirements; (6) maintenance and modification--the continual repetition of phases 1 through 5 as the system evolves in time to meet new organizational needs.

It is clear from available data that the life cycle process is dynamic in nature. One phase is not completed before another one begins. Each of the phases overlaps and interacts. Part of the interaction is due to the natural state of affairs. A requirements change or flaw, or better understanding of what was meant in an earlier phase, causes a return to that phase with respect to the particular subsystem currently being worked on.

Many methodologies involve the interaction of the various phases of development. For example, the top down development methodology permits certain parts of the system to be coded before other parts are designed (Mills, Baker). Methodologies exist that require an iterative development of the system by building increasingly larger subsets of the system until the entire functional capability is developed (Basili and Turner).

Standard static milestones become meaningless in these environments because they tend to be too simplistic a measure of progress. A milestone should be a measurable checkpoint of progress. It must be a visible, certifiable, if not quantifiable, measure related to an individual phase and subsystem with respect to degree of completeness; i.e., satisfaction of functional goals, bounds on reliability, "goodness" of the product, etc. The milestone should assure us that a particular function has been completed through some phase with a certain reliability and complexity bounds, etc. This permits a more honest evaluation of progress.

Certifiable milestones are a necessity for successful management. Milestones can be evaluated by nonquantifiable and quantifiable techniques. Nonquantifiable evaluation involves reading and reviewing the product at various stages in development. Quantifiable evaluation implies the need for measures of the existing product, whether that product is code or some design or specification or requirement. A simple measure of lines of code is not enough. It only measures the final product. This kind of measurement is too late to give any information about progress when it is really needed and when something can be done about problems. It does not tell us enough. We want more than lines of code. We want to know how complex that code is, how reliable it is, how transportable it is, etc. We want to know something about the quality of the product at that point in time.

Life cycle management requires an automated set of tools to help the manager evaluate progress, evaluate the product, and understand when and if corrective measures need to be made.

Model and Measure Development and Technology Transfer

Models and measures tell us what is supposed to be happening, what we need to know, and how to recognize it. They drive the actual data collection process and require the standardization of definitions and taxonomies. To develop a realistic model, we must better understand the process. To understand the process, we must have a realistic model. The approach is circular, but we must start somewhere. We can begin by building a set of "first-order" models and measures based on sound principles and experience. These models and measures then can be used to generate the collection of data that will help verify this validity. Based on the data collected, we refine the models and possibly modify the kinds of data collected. The process is iterative and continues until we have a set of models and measures that corresponds to reality and tells us what we want to know.

Although the overall approach is slow in yielding a full understanding of the entire life cycle process, intermediate models and measures are still valuable for management in the present environment. There are several models and measures that already exist and have shown tremendous promise of realistically mirroring various aspects of the life cycle. These models can be used to aid in the management of current projects and help in the problem of technology transfer.

There are essentially two kinds of technology transfer. The first is the transfer of theory into practice; the second is the transfer of knowledge and experience from one project to another. Both require the collection of data and the use of tools which feed back and save information on the management and development of the system. Some data collection must be done manually, but much of it may be automated. The use of automated tools can help standardize definitions, aid in the development of taxonomies within the local environment, and minimize the cost of data collection. These tools can be used to feed back information to the manager in real time, permitting him to control the development of the product. Some of this information can also be of benefit to the programmer, permitting him to evaluate his own progress or evaluate the complexity or reliability of the design or code he has developed. Information gathered during the development of one project can be stored in an archival data base which the organization can use to do future estimation on costs, development time, etc. It can be used after the fact to evaluate what went wrong and what went right. Other managers can use it as an experience base to better understand how different projects with different parameters were managed. The feedback can be used to help evaluate some new methodology, or help modify or adapt it to the local environment. It can be used to convince managers of the benefits of a different methodology by giving real evidence that it works.

This archival data base can then be used to validate and refine existing models and measures with respect to individual environments. It may also motivate new theories and measures or the collection of new data; i.e., we believe something went wrong because we didn't understand the effect of some parameter which we had not measured.

Specific Models and Measures

There currently exists a large number of models and measures of the software development process. Many of them are referenced in the working papers of the First Software Life Cycle Management Workshop (1 SLCMW) and in the proceedings of the Second Software Life Cycle Management Workshop (2 SLCMW). These range from resource estimates to complexity measures and we will discuss just a few here to demonstrate the amount of progress in the area.

1. Resource estimation - Results from computer hardware estimation have been used for the basis of software estimation (Putnam). The expenditures across time for large-scale projects have been found to agree closely with the Rayleigh curve

$$E = 2\ Kate^{-at^2}$$

where E is the rate of expenditure at time period t (measured in units such a month, year), K is the total cost of the project, and a is the maximum expenditure for any time unit. Further work has been done on analyzing the equation to add such factors as difficulty of the product and state of the art of technology. The model has shown to correspond fairly well to any large-scale model developments. Various other models have been developed based on experience, data collection, and continual refinements (Boehm and Wolverton).

2. Effect of various factors - Effect of various factors and development techniques on the software environment have been measured (Walston and Felix) yielding a productivity ratio of

$$P = 5.2\ L^{0.91}$$

where P is the total effort in man months and L is in thousands of lines of source code. A productivity index has been derived which weights many of the factors to identify divergence from the estimated value P. These factors include such items as complexity of customer interface, hardware under concurrent development, previous experience of personnel with the application, programming language and operational computer, techniques used such as structured programming, chief programmer teams, top down development, and complexity of the application and program flow.

3. Reliability models - A variety of reliability models have been developed (Musa, Shooman, Littlewood, Goel), several of which have shown to be quite effective in practice in estimating the amount of machine time required to reach a certain predetermined reliability standard in order to stop testing the system. One concept used is mean time to failure.

4. Product measures - Several measures of the complexity of a product have been developed which can be associated with the control structure (McCabe) or data structure (Myers). Development invariants in the software product with regard to such factors as effort and length estimation have been identified (Halstead). Invariants in the behavior of a product in the software maintenance have also been identified (Belady and Lehman). A variety of measures, including portability, reusability and maintainability, have also been developed (McCall).

Tools have been developed that lend themselves to automatic data collection, visibility and measurement. These include various requirement analyzers, such as PSL/PSA (Teichroew and Sayani) and automated Process Design Languages (Caine and Gordon).

Conclusion

It is clear that to manage software we need to better understand the software life cycle process. To this end, data collection and analysis are required to build and refine models and measures of the process and the product. There is a need for identification of influencing factors, standard definitions and taxonomies, certifiable milestones and tools that aid in the data collection analysis and feedback process. Progress has been made in the modeling of many aspects of the life cycle, but more testing of these models must be done in real environments. The process is slow but necessary if we are ever to truly understand the software life cycle process.

References

(Mills)
Mills, Harlan D., Software Development, IEEE Transactions on Software Engineering, Vol. SE-2, No. 4, December 1976, pp. 265-273.

(Baker)
Baker, F. T., Structured Programming in a Production Programming Environment, Vol. SE-1, No. 2, June 1975, pp. 241-252.

(Basili and Turner)
Basili, Victor and Turner, A. J., "Iterative Enhancement: A Practical Technique for Software Development, IEEE Transactions on Software Engineering, Vol. 1, December 1975, pp. 390-396.

(1 SLCMW)
Software Phenomenology, Working Papers of the Software Life Cycle Management Workshop, Airlie House, August 21-22, 1977

(2 SLCMW)
Proceedings of the Second Software Life Cycle Management Workshop, Atlanta, Georgia, August 1978, IEEE Society Publication.

(Putnam)
Putnam, Lawrence H., A General Empirical Solution to the Macro Software Sizing and Estimating Problem, IEEE Transactions on Software Engineering, Vol. SE-4, No. 4, July 1978, pp. 345-361.

(Boehm and Wolverton)
Boehm, B. W. and Wolverton, R. W., Software Cost Modeling: Some Lessons Learned, Proceedings of the Second Software Life Cycle Management Workshop, Atlanta, Georgia, August 1978, IEEE Society Publication.

(Walston and Felix)
Walston, C. E. and Felix, C. P., A Method of Programming Measurement and Estimation, IBM Systems Journal, No. 1, 1977

(Musa)
Musa, J. D., A Theory of Software Reliability and Its Application, IEEE Transactions on Software Engineering, Vol. SE-1, No. 3, pp. 312-327, September 1975.

(Shooman)
Shooman, Martin L., Structural Models for Software Reliability Prediction, Proceeding of the 2nd International Conference on Software Engineering, October 1976, San Francisco, California, IEEE Computer Society, New York

(Littlewood)

Littlewood, Bev, Validation of A Software Reliability Model, Proceedings of the Second Software Life Cycle Management Workshop, Atlanta, Georgia, August 1978, IEEE Society Publication.

(Goel)

Goel, Amrit L., A Software Error Detection Model with Applications, Proceedings of the Second Software Life Cycle Management Workshop, Atlanta, Georgia, August 1978, IEEE Society Publication.

(McCabe)

McCabe, Thomas J., A Complexity Measure, IEEE Transactions on Software Engineering, Vol. SE-2, No. 4, December 1976, pp. 308-320

(Myers)

Myers, G. J., Reliable Software Through Composite Design, Petrocelli/Charter, 1975

(Halstead)

Halstead, M, Elements of Software Science, Elsevier Computer Science Library, 1977

(Belady and Lehman)

Belady, L. A. and Lehman, M. M., A Model of Large Program Development, IBM Systems Journal, No. 3, 1976, pp. 225-251.

(McCall)

McCall, James A, The Utility of Software Quality Metrics in Large-Scale Software System Developments, Proceedings of the Second Software Life Cycle Management Workshop, Atlanta, Georgia, August 1978, IEEE Society Publication.

(Teichroew and Sayani)

Teichroew, D. and Sayani, H., Automation of System Building, Datamation, pp. 25-30, August 15, 1971

(Caine and Gordon)

Caine, S. H. and Gordon, E. K., PDL: A Tool for Software Design, Proceedings 1975 National Computer Conference, pp. 271-276.

Contingency Theory Approach to Systems Life Cycle Management

J. David Naumann
Gordon B. Davis

University of Minnesota

Application of formal life-cycle methodology in an organizational response to an information-decision problem, especially the reduction of uncertainty about the outcome of the development process. A single life-cycle methodology is inappropriate because the level of uncertainty varies as a function of several contingencies.

This paper described contingencies which determine the level of uncertainty, described a continuum of responses to level of uncertainty, and discussed the application and testing of the contingencies theory. Contingencies include project size, degree of structuredness, user-task comprehension, and developer-task proficiency.

THE CONTINGENCY THEORY APPROACH TO SYSTEM LIFE CYCLE MANAGEMENT

J. David Naumann and Gordon B. Davis

University of Minnesota
College of Business Administration
Minneapolis, Minnesota

Application of formal life-cycle methodology is an organizational response to an information-decision problem, specifically the reduction of uncertainty about the outcomes of the development process. A single life-cycle method is not appropriate because the level of uncertainty varies as a function of a number of contingencies. This paper describes contingencies which determine the level of uncertainty, describes a continuum of responses to level of uncertainty, and discusses the application and testing of the contingency theory. Contingencies include project size, degree of structuredness, user-task comprehension, and developer-task proficiency.

The system development life-cycle is the central concept in currently-used methods of managing and controlling the determination of information requirements and designing and implementing processing systems to meet those requirements. When organizations specify the use of formal life cycle-based methods for all application developments, the results are mixed. A single life-cycle method is not appropriate for all cases because applications differ in the certainty with which requirements can be established.

Introduction

Information systems developers who rely upon formal life-cycle development methodologies are not universally successful; developers who do not apply life cycle methodologies do not always fail. Formal life-cycle management procedures, rigorously adhered to, provide a high degree of assurance of success for large, conventional developments where users and developers share understanding of the results to be produced. As systems under development move away from the routine, however, uncertainty about the characteristics of the end result increases. Information requirements determination and specification becomes more difficult, costly, lengthy, and unsatisfying to users and developers alike.

Concepts such as recursive life cycle[1] and prototype systems[2] are receiving increasing attention as suggested solutions to the problems of information requirements determination. Implicit in the advocacy and acceptance of these concepts is an understanding that the development and application of increasingly rigid rules and programs for requirements determination and specification can be counterproductive. Like the formal life-cycle methodology, however, an innovative concept is not a panacea. Different methods may be needed for different systems development projects.

A contingency theory identifies alternative actions and presents factors to use in selecting the optimal alternative. For example, McFarlan[3] proposes a contingency theory for development project management. He identifies project size, degree of structuredness, and degree of company-relative technology as factors which determine the best project planning and control tools.

Uncertainty has been identified by Galbraith[4] as a major factor in determining the optimal organization structure. The difference between the amount of information necessary to perform a task and the amount of information possessed is a measure of task uncertainty; organizations respond by choosing from a set of four organizational strategies to deal with the level of uncertainty.

The information requirements determination and specification problem is one of uncertainty: the success of an information system development effort depends upon the clear, complete, unambiguous, and accepted specification of need. The formal life-cycle development methodology response to this need is to require formally approved documentation as an agreement between users and developers before design and implementation. Ideally, this approach reduces uncertainty about the outcome of the development process to a mutually acceptable level. Since information requirements uncertainty varies from application to application, the "rules and programs" approach is incapable of producing a satisfactory result over the entire range of information systems development projects.

Contingencies Analysis

In the determination of information requirements for an information system application uncertainty refers to knowledge of the "real" information needs. Among the development contingencies which determine information requirements uncertainty are project size, degree of structuredness, user-task comprehension, and developer-task proficiency. A systems development project has some combination of these attributes. The combination of contingencies defines the level of uncertainty which must be resolved in a system development effort.

Project Size

The project size contingency has three key characteristics: duration, number of people involved, and total dollar amount. These characteristics are usually, but not necessarily, collinear. That is, a high cost project usually requires many people over an extended time period. Project size is not a good measure of the value of a systems development project, but it is correlated with the degree of uncertainty of the results of the development process.

The number of people involved characteristic refers to both users and system developers. A large project necessarily involves many developers, an extended duration, or both. Uncertainty is associated with communication and coordination in large development projects. A project with a large number of users also contributes to uncertainty. Multiple users or even multiple user organizations add to the information which must be gathered and communicated to determine system requirements.

Degree of Structuredness

One dimension of the Gorry and Scott Morton[5] framework for information systems is that of the relative structuredness of the decisions to be supported by an information system. For information systems information requirements determination, a high degree of structuredness means that a general model exists which can be applied to the given organizational setting. A low degree of structuredness means that there is no routine procedure for dealing with the problem, there is ambiguity in the problem definition and uncertainty as to the criterion for evaluating solutions. Uncertainty about the decisions to be supported is an important factor in uncertainty about the outcome of the systems development process.

User Task Comprehension

Related to but distinct from structuredness is the comprehension that the user or users have of the task to be performed by the information system. User task comprehension affects the selection strategy and development project success in much the same way as degree of structuredness. If the users have a low degree of understanding of the task for which the system is intended, whether or not a general model of a problem exists, less is certain about the information requirements (and the users' acceptance of the results of the development process).

Developer Task Proficiency

Developer task proficiency is a measure of the specific training and experience brought to the project by the development staff: project manager, liaison staff, systems analysts, systems designers, programmers, etc. It is not a measure of ability or potential: rather it is a measure of directly applicable experience. This contingency indicates the degree of uncertainty with which the developer will be able to obtain and document the requirements (and also proceed with the remainder of the development process).

CONTINGENCY ANALYSIS		
CONTINGENCY		CONTRIBUTION TO UNCERTAINTY
TYPE	DEGREE	
Project Size	Large Small	+ -
Degree of Structured-ness	Structured Unstructured	- +
User Task Comprehension	Complete Slight	- +
Developer-Task Proficiency	High Low	- +

LEVEL OF UNCERTAINTY

Figure 1

INFORMATION REQUIREMENTS DETERMINATION CONTINGENCY ANALYSIS

Uncertainty-Reducing Strategies

The response to uncertainty produced by the characteristics of a systems development task, the using organization, and the developer organization (i.e., the contingencies) has frequently been unidimensional. Under the traditional life-cycle approach, formal procedures, reviews, committees, check points, etc., are used for all projects.[6] There has been no recognition of the degree of uncertainty from the contingencies. An alternative approach is to:

1. Analyze contingencies and determine the relative uncertainty,
2. Select an uncertainty reducing strategy appropriate to the level of uncertainty,
3. Apply the information requirements determination methodology corresponding to the appropriate uncertainty reducing strategy.

A low level of uncertainty clearly suggests that a simple strategy will suffice to discover information requirements and produce system specifications. A high level of uncertainty, in contrast, suggests that the appropriate strategy will serve to define and communicate information requirements to users and developers in such a way that the object system products can be specified and agreed to.

The strategies range from acceptance of information re.uirements as specified, through linear discovery and recursive discovery, to experimental discovery of information requirements. Corresponding methodologies are suggested as typical of the discovery strategies and widely understood and applicable.

Accept as Specified

If information requirements are known and agreed upon, then the proper strategy is to accept the user's statement of need as adequate specification for implementation. The method is therefore to have no information requirements cycle. Examples are file conversions, reports from existing files or databases and small single-user models. These examples have in common: small size, high degree of structure, users who understand what the systems are to do and how the implementation will function, and must have developers with experience in similar systems. Explicit recognition of the need for the "accept as specified" strategy will lead to greater responsiveness and an increase in development organization efficiency. Formal rules and procedures designed to assure mutual understanding and acceptance of system specifications may be unnecessary and unwieldy at this level.

Linear Discovery

If information requirements can be determined through a straight-forward process of interviewing, fact gathering, and documentation, the proper strategy is to proceed step-by-step to system specification. The method is therefore a linear application of the life cycle. Examples are transaction level systems, single function accounting systems such as accounts receivable or payable, and minor modifications to existing information systems.

Information requirements for large systems which are highly structured and where user-task comprehension and developer-task proficiency are high may be effectively determined by the linear discover process. However, information requirements for a relatively small system such as a decision model may not be determinable by this method if the decisions to be supported are relatively unstructured, or if the user does not comprehend the task, or if the developers have not previously produced such a system. Linear application of the life-cycle model is an effective strategy under the appropriate combination of contingencies.

Recursive Discovery

The linear discovery strategy may not produce correct or complete or acceptable specifications of information system requirements. The traditional life-cycle approach extends to recursion for such systems. One or more discovery tasks are iterated until a complete, consistent specification is determined and accepted. Examples are large, multiple-user systems, systems which are new to the user or developer organization, and systems which support the relatively unstructured decisions of tactical and strategic management. This approach assumes that a correct specification of requirements can be made given sufficient time and effort. Where the contingencies indicate that is a valid assumption, the recursive discovery strategy is appropriate and effective.

Experimental Discovery

A high level of uncertainty may be indicated by a combination of the contingencies. Repeated iterations of discovery may not successfully produce adequate specifications of information requirements in such cases. The life cycle method, whether linear or recursive, is inappropriate when uncertainty is high. The strategy of experimental discovery as realized in the prototype design method, reduces uncertainty by producing successive approximations.

Users and developers can easily see what is wrong with an implementation even though they are unable to completely specify its information requirements.[7] The higher costs associated with prototype implementation are justified by the provisions for interactive development and discovery. Examples are decision support systems for upper management, interactive forecasting models, and small (or large) systems to be implemented for many different users. Conscious selection of the experimental discovery strategy may be the only effective approach to information requirements determination when the level of uncertainty is high.

	INFORMATION REQUIREMENTS DETERMINATION	
	Uncertainty Reducing Strategy	Methodology
low UNCERTAINTY	Accept information requirements as specified	No requirements determination
	Linear information requirements discovery	Life cycle applied linearly
	Recursive information requirements discovery	Life cycle applied recursively
high	Experimental information requirements discovery	Prototype development

Figure 2

INFORMATION SYSTEMS DEVELOPMENT STRATEGY AND METHODOLOGY

Conclusion

A range of information requirements determination strategies is needed. Such strategies match the level of uncertainty about the system specification which is to result. The appropriate strategy is determined by the extant contingencies.

Inappropriate methodology selection and application leads to either of two problems: insufficient capacity to reduce uncertainty or more capacity than needed. Where a strategy does not provide sufficient capacity to reduce uncertainty, several possible consequences may result. Changes during implementation or after installation are often required. User dissatisfaction with the object system is a negative result even though modifications are not made. In the extreme case, information requirements must be re-analyzed and specified.

The problems caused by application of a higher capacity methodology than needed are less immediately apparent. Higher development costs and longer development duration reduce the efficiency of the development organization and justify the charge of unresponsiveness.

Uncertainty reducing strategies and a wide range of information requirements determination methodologies are being applied in industry and government. A program of empirical research is needed to develop and refine measures of uncertainty indicated by the contingencies to appropriately characterize specific methodologies as specific levels of uncertainty reducing strategies, and to associate systems development outcomes with the strategies applied.

The contingency theory approach is applicable to another aspect of information systems development which might be labeled the uncertainty tolerance level. Variables such as the impact of systems failure and the cost of modification and enhancement imply that uncertainty need not be reduced to a common level for all systems, but that an appropriate level of uncertainty is contingent upon such factors as number of users, system distribution, etc.

Research leading to more precise operational definition of the contingencies and their effects will lead to more efficient and effective information systems development.

References

1. J. L. Podolsky, "Horace Builds a Cycle," Datamation, November, 1977, page 162.

2. L. Bally, J. Britton, and K. H. Wagner, "A Prototype Approach to Information Systems Design and Development," Information Management, Volume 1, Number 1, November, 1977.

3. F. W. McFarlan, "Effective EDP Project Management," in Managing the Data Resource Function (R. Nolan, Ed.), West Publishing Company, St. Paul, 1974.

4. J. Galbraith, Designing Complex Organizations, Addison-Wesley, Reading, Mass., 1973.

5. G. A. Gorry and M. S. Scott Morton, "A Framework for Management Information Systems," Sloan Management Review, Volume 13, Number 1, (Fall, 1971), pages 55-70.

6. G. B. Davis, Management Information Systems: Conceptual Foundations, Structures, and Development, McGraw-Hill, New York, 1974.

7. C. Alexander, Notes on the Synthesis of Form, Harvard University Press, Cambridge, Mass., 1964.

OPERATING SYSTEMS SECURITY

LTC Robert P. Campbell
DAMI-AM

OPERATING SYSTEMS SECURITY

SESSION CHAIRPERSON: LTC Robert P. Campbell

DAMI-AM

SESSION SUMMARY

This session featured presentation of two papers which briefly summarized the character and scope of the software security problem, discussed approaches being used to improve the security of general purpose operating systems and described in detail current efforts on development of the Department of Defense Kernelized Security Operating System (KSOS). The KSOS presentation detailed the design methodology and security assurance methods used, the actual design, with emphasis on the interfaces available to various classes of users; and the potential application for KSOS.

SOFTWARE SECURITY

LTC Robert P. Campbell
OACSI, HQDA

The problems of ensuring the security of software and the operational systems which they support have been with us, naggingly, for over a decade. A basic conflict can be found between the national level security policies for the protection of classified information and privacy data, which are not enforceable within today's automation technology, and operational necessity, pressuring for full exploitation of state-of-the-art technology (Figure 1). These factors, considered with technology, form a triumvirate within which those with security responsibilities must achieve an equilibrium, a balance, a degree of risk that is acceptable within the context of the environment. The policies form the base line. They are generally fixed and immutable or at best can only be more precisely defined and thus refined. On the other hand, there are requirements, born of operational necessity, driving us to employ all available technology in satisfaction of these needs. The third element, technology, responds to both policy and operational requirements. A change in one of these areas influences the other two.

Within this environment, the price of security is very high. Because technology has not been able to respond to security policy needs, sizeable amounts of resources have been allocated to support information segregation, dedicated computers, large numbers of security clearances, and burdensome physical precautions and procedures. There are other nondirect costs, in terms of inconvenience, lack of capability, or lost opportunity that are never really quantified but which clearly exist. The point is that large quantities of money and other resources are being put into day-to-day operational procedures without materially influencing the problem. These are sunk cost which are lost, never to be retrieved, and which drastically reduce the return on our technological investment.

These sunk costs are increasing. It was estimated that, 10 years ago, the commercial world found a premium of 3 to 5 percent for security to be too high. Today, there is a belief that, with the increased emphasis being given to security and privacy, there is a willingness to raise this premium to 10 to 15 percent, a penalty that is bound to impact broadly upon overall operations. It is evident that the time has come to directly apply some of our technological drive to satisfaction of these privacy and security requirements.

There is an inherent compatibility between our quest for reliable software and that for secure software. Reliability seeks assurance that the implementation does what it is supposed to do, while security seeks assurance that the implementation not only does what it is supposed to do -- but no more or no less. While reliable operations are not necessarily secure, secure operations always have the characteristic of reliability because security requires a high degree of stability and predictability, both of which are reliability prerequisites. There is growing awareness that the time has come to invest in security technology in order to offset the increasing burden of operational security costs. Progress needs to be made.

Shifting now to the most basic of our software security problems, that of the operating system itself, there are basically three strategies that can be used to address security vulnerabilities: the patch method, the security kernel method, and that of designing a secure operating system from scratch (Figure 2). The patch method has been characterized by such attempts as by IBM to improve its 360/370 security with VS2/Rel 3, the CDC 6600 with NOS BE, the Honeywell GCOS, the UNIVAC Exec VIII, and so on. The conclusion, quickly learned, is that security by patch does not work. There are always other holes, and often the patch itself will introduce more flaws. The second method features the security kernel, wherein the trusted processes are isolated in a special module that checks all accesses to the system. This method shows great promise for short-term improvement in security state-of-the-art. There are security kernels being developed for the general purpose operating systems of the PDP-11 (the "KSOS 11"), the Honeywell Level 6 (KSOS 6), and the IBM VM 370 (KVM 370). Some of these implementations will reach operational capability before the end of FY 79. The third strategy, that of designing a secure operating system from scratch, offers the best potential for long term solution although it will not be without problems. Anytime you introduce a totally new operating system, you will not have the support base, the community of users or a broad inventory of applications software readily available for use. But DOD is pressing ahead with the Provably Secure Operating System, or PSOS, looking for a prototype by 1980, with the first formal, mathematically verifiable version possibly available by late 1982. The National Security Agency, Air Force Systems Command, SRI, and MITRE are currently collaborating on the design effort.

With the promise being shown by the security kernel approach, we believe that we are on the brink of significant technical accomplishment. To maximize the effectiveness of these efforts, DOD has established a Computer Security Technical Consortium (Figure 3) in order to provide (1) a coordination mechanism for ongoing research in the computer security field, (2) a focus for technical aspects of the certification and implementation of multilevel secure systems, and (3) technical leadership for the transfer of state-of-the-art computer security technology to industry.

The third area of this initiative involves an industry relations program which will transfer this technology to industry and encourage industry, at its own expense, to develop and implement secure systems based upon this technology. To further this effort, DOD has asked the military services for funding to equally support nominal contracts with computer manufacturers to obtain information on the integrity mechanisms they are using in their new operating systems. DOD has also requested technical support in the form of knowledgeable individuals to act as Contracting Officers Technical Representative (COTR) for each of the industry contracts. These individuals must be government employees and will be expected to devote approximately 20 percent of their normal duty to this effort. I believe the Army's need for multilevel secure systems to be as great, if not greater, than that of any of the services, so I am currently trying to line up Army support for this effort.

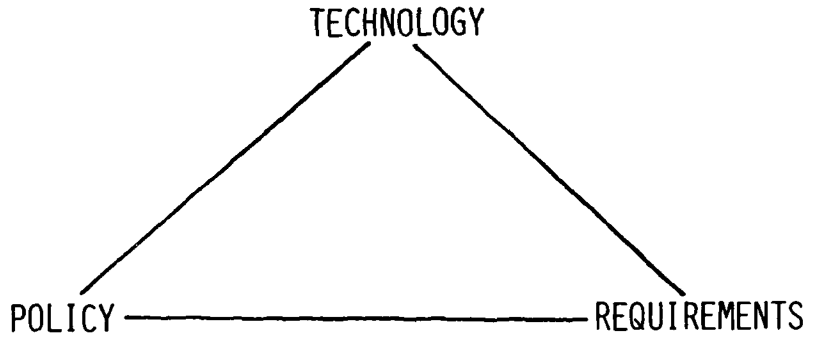
The remainder of this session will focus on the Kernelized Secure Operating System, or KSOS, as it is currently entitled (Figure 4). The KSOS had its beginnings in research that started almost 10 years ago when concerns for the insecurity of sensitive computer systems led to the use of "tiger teams" to attempt penetration through their operating systems. These teams succeeded in breaking every commercially available operating system with ease and impunity. Concern over the serious vulnerabilities uncovered led to a study in 1972 by the Air Force Electronic Systems Division of the requirements for a secure system. The results of that study form the basis of most of our efforts in developing secure systems. The most important concept to come out of that study is that of the reference monitor (Figure 5). The reference monitor mediates the access of subjects to objects. It determines whether or not access is to be granted. All security relevant decisionmaking code is collected in the security kernel. The reference monitor concept requires that this module be complete (i.e., that all subject/object accesses be checked by the kernel), that it be isolated (i.e., that the kernel code be protected from modification or interference by any other code in the system), and that it be verifiable (i.e., perform only that which it was intended and no more). Verification is a very difficult area and may run the gamut from mere performance audit to formal mathematical proof of correction. For security applications, we are striving for formal provability. Significant progress is being made in this area. Between 1973-1975, the Air Force ESD and MITRE developed a security kernel for the PDP 11/45 and then applied those concepts to the design of the Multiplexed Information and Computing Service (MULTICS) operating system for the Honeywell 6000 system. About this same time, Dr. Jerry Popek of UCLA was building a secure kernel prototype for the PDP 11/45 based upon a virtual machine monitor system and emphasizing the kernel verification process. Also during the early 1970's, the Bell Labs designed and implemented the UNIX operating system for the PDP 11. The UNIX, designed to provide effective support to interactive users, has proven to be highly efficient and reliable. Its operating system structure, written in the high order language "C" is simple and uncomplicated. Both UCLA and MITRE have interfaced their prototypes with UNIX.

In 1977, DOD initiated an effort to move beyond the prototype stage, to design and implement a production quality version of a certifiably secure operating system which emulates the Bell Lab's UNIX. This program, known at that time as the "DOD Secure UNIX," was two-phased (Figure 6). The Design Phase, with a competitive PFP issued in April 1977, saw twc contractors (TRW and Ford Aerospace and Communications Corporation) selected in August 1977 to develop detailed systems designs. The Design Phase was completed in April of 1978. The Implementation Phase commenced in May 1978 with the award of contract to Ford Aerospace and Communications Corporation to build the production version, fully supported, for fielding i. August 1979.

KSOS will be a complete rewrite of the core of the UNIX operating system (Figure 7). Thus, there will be no Bell Lab's/Western Electric code in the KSOS and no need to pay licensing fees to Western Electric. Through provided emulation, however, KSOS will be compatible with the UNIX operating system and existing UNIX applications.

Because of the simplicity and straightforward design of UNIX, extending the KSOS to run on other hardware architectures should not be a difficult task. The UNIX itself currently runs on PDP 11/40, 11/45, 11/70, and also on Interdata 8/32. UNIX minicomputer applications are also frontending IBM 360/370's, UNIVAC, and Honeywell mainframes, testifying to the appeal and acceptability of UNIX. Al! of these factors favor the DOD initiative. In addition to DEC and Honeywell implementations of KSOS, DOD is aware of other serious interests in implementing KSOS-like architectures. There is no reason to believe that other major manufacturers will not do likewise. The KSOS, about to join the DOD inventory of computer tools, will provide a significant capability to better respond to national level security policy and, importantly, will be demonstrated proof to industry that secure systems are possible.

AUTOMATION SECURITY
THE "ETERNAL TRIANGLE"



TECHNOLOGY

POLICY

REQUIREMENTS

Fig. 1

SECURE OPERATING SYSTEM STRATEGIES

- PATCH
    - IBM VS2/REL3
    - CDC 6600 (SCOPE/NOS BE)
    - HIS 6000 GCOS
- SECURITY KERNEL
    - KSOS 11
    - KSOS 6
    - KVM 370
- ORIGINAL DESIGN
    - PSOS

Fig. 2

DOD COMPUTER SECURITY
TECHNICAL CONSORTIUM

- TO COORDINATE ONGOING RESEARCH
- TO DEFINE THE TECHNICAL ASPECTS OF CERTIFICATION AND IMPLEMENTATION OF MULTILEVEL SECURE SYSTEMS
- TO PROVIDE TECHNICAL LEADERSHIP FOR TRANSFER OF TECHNOLOGY TO INDUSTRY

Fig. 3

EVOLUTION OF THE DOD
KERNELIZED SECURE OPERATING SYSTEM

1968-1973: "TIGER TEAM" PENETRATIONS

1972: USAF/ESD STUDY - REFERENCE MONITOR CONCEPT

1973-1975: ESD/MITRE PDP 11/45 SECURITY KERNEL PROTOTYPES

1974-1975: UCLA SECURITY KERNEL/VIRTUAL MACHINE MONITOR - PDP 11/45

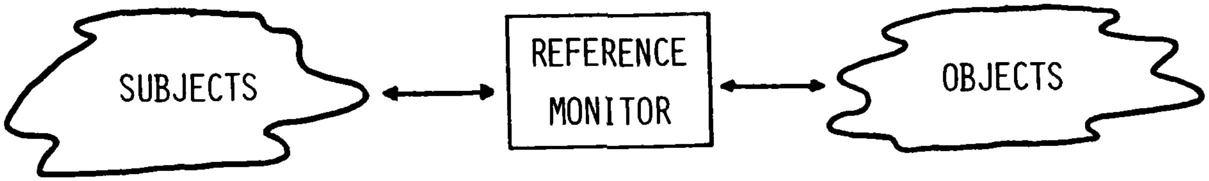
1971-1975: BELL LABS UNIX DEVELOPED

1976: UCLA AND MITRE BEGIN SECURE UNIX PROTOTYPES

1977: DOD "SECURE UNIX" PRODUCTION VERSION EFFORT INITIATED

Fig. 4

REFERENCE MONITOR CONCEPT



- COMPLETE
- ISOLATED
- VERIFIED

Fig. 5

DOD KERNELIZED SECURE OPERATING SYSTEM (KSOS)

o DESIGN PHASE

APR 1977: COMPETITIVE RFP

AUG 1977: DESIGN CONTRACTS AWARDED

APR 1978: DESIGN PHASE COMPLETE

o IMPLEMENTATION PHASE

MAY 1978: CONTRACT AWARDED

AUG 1979: IMPLEMENTATION COMPLETE

Fig. 6

KSOS

- COMPLETE REWRITE OF UNIX CORE
- NO WESTERN ELECTRIC CODE IN KSOS
- COMPATIBILITY WITH EXISTING UNIX ENVIRONMENTS

Fig. 7

The Department of Defense Kernelized Secure Operating System (KSOS)

E. J. McCauley

Ford Aerospace and Communications Corporation

The Department of Defense Kernelized Secure Operating System (KSOS) is intended to be a provably secure operating system for larger minicomputers. KSOS is divided into three parts:

1. The Security Kernel - a minimally complete primitive operating system providing the basic security enforcement of the system.

2. The UNIX** Emulator - which transforms the interface provided by the Kernel into one similar to that provided by UNIX. Existing UNIX applications will run unmodified on KSOS.

3. The Non-Kernel System Software (also called Non-Kernel Security-Related Software) - which is the collection of software needed to operate, maintain, and administer a KSOS system.

The session dealt with three major topics. First, the design methodology and security assurance methods used on the KSOS project have been briefly discussed. These methods blend well-accepted software design procedures with new ideas from the research community. KSOS will be one of the first production projects to routinely employ formal, mathematical specifications in its design process. There were also proofs that the design satisfies the security requirements independent of any mechanization of that design. The second topic was the design of KSOS. Here, the emphasis was on the interfaces available to various classes of KSOS users. It has been a design goal that the KSOS Kernel could be used for applications other than UNIX. Thus, the features that realize this goal were emphasized. The final topic was the potential applications for KSOS. Particular emphasis was placed on applications in which the full generality of the UNIX Emulator is not required, such as the Military Message Processing equipment.

The Department of Defense Kernelized Secure Operating System (KSOS) *

E.J. McCauley

Ford Aerospace and Communications Corporation, Palo Alto, CA

ABSTRACT

The Department of Defense Kernelized Secure Operating System (KSOS) is intended to be a provably secure operating system for larger minicomputers. KSOS will emulate the UNIX** operating system. This paper deals with three major topics. First, the design methodology and security assurance methods used on the KSOS project will be briefly discussed. These methods blend well-accepted software design procedures with new ideas from the research community. KSOS is one of the first production projects to routinely employ formal, mathematical specifications in its design process. There will also be proofs that the design satisfies the security requirements independent of any mechanization of that design. The second topic is the design of KSOS. Here the emphasis will be on the interfaces available to various classes of KSOS users. It has been a design goal that the KSOS Kernel could be used for applications other than UNIX emulation. Thus, the features that realize this goal will be emphasized. The final topic will be the potential applications for KSOS. Particular emphasis will be placed on the effective utilization of the system's features in supporting multi-level applications.

1. INTRODUCTION

The purpose of the Department of Defense Kernelized Secure Operating System (KSOS, formerly called Secure UNIX) is to provide a provably secure operating system for larger minicomputers. KSOS will provide a system call interface closely compatible with the UNIX operating system. The initial implementation of KSOS will be on a Digital Equipment Corp. PDP-11/70 computer system. A group from Honeywell is also proceeding with an implementation for a modified version of the Honeywell Level 6 computer system.

KSOS will be capable of handling information at various security levels (a security level is a combination of a hierarchically ordered classification category, like SECRET or TOP SECRET, and a, possibly null, set of compartments, like NOFORN or specialized need-to-know compartments). The goal of the system is to provide strong assurances that it is impossible for an unprivileged user to cause an information compromise.

* The work described in this paper was performed under ARPA Order 3319, Contract MDA903-77-C-0333 administered by the Defense Supply Service-Washington. Various DoD Agencies are funding the work. The conclusions presented are those of the author and are not necessarily those of the Government or Ford Aerospace.

** UNIX and PWB/UNIX are trademarks of the Bell System

At its outer interface, KSOS will appear to be closely similar to the UNIX operating system [Ritchie 74]. The only changes are to tighten the security checking on some of the operating system calls, and to add several new calls which individual UNIX sites had previously added to their systems. Existing applications programs written for UNIX will run without modification or recompilation on KSOS, providing that they do not violate the security rules of the system. At last count there were several hundred application programs for UNIX, ranging from simple utilities through sophisticated compilers, data management systems, text processing systems, and powerful editors. (This paper was completely prepared on a UNIX system, as is all documentation for the KSOS project.) All of these programs should run on KSOS without modification.

This UNIX-like interface is provided by a software component called the UNIX Emulator. The UNIX Emulator transforms the user's UNIX operating system calls into (sequences of) calls to the Security Kernel. The Security Kernel is the heart of the system. The Kernel implements the reference monitor concept [Bell and LaPadula 73]. Briefly, through a combination of hardware and software checking, the Kernel monitors every access attempt by each user process. The Kernel will be shown to make the correct decision on whether to permit or deny the access attempt.

One important distinguishing characteristic of KSOS over the prototypes which have preceded it [Kampe et al. 77] [MITRE 77] is that it contains a full range of support software. Included in this "Non-Kernel System Software" (also called Non-Kernel Security-Related Software) are components which support the day-to-day operational functions of the system: secure spooling of line printer output, portions of the interface to a packet-switched computer network, etc. Also included are components for the continuing maintenance of the system such as consistency checks of the file system, and system generation support. Finally, there are components to support the administration of the system, such as adding and deleting users, changing the security levels that a given user may access, and other functions.

The schedule for KSOS calls for its delivery in the fall of 1979 after the conclusion of a full series of testing. The KSOS development contract specifies that the system shall have a full MIL SPEC documentation package in accordance with MIL-STD-483, -490, and -1521A. Detailed documentation on the basic architecture and interfaces is presently available in the form of B5 Specifications. The Kernel B5 Specifications [Kernel 78] include formal, mathematical descriptions of precisely what each Kernel call does. In addition technical reports have been delivered detailing our plans for verification of the system's security properties [Verif 78], for the tools and techniques to be used in implementation [Impl 78], and for the long term maintenance and support of the system [Maint 78].

2. THE DESIGN METHODOLOGY

It is generally accepted that security cannot be added on to a system. Rather, the security features of a system must be designed in from the beginning. In the case of KSOS this is accomplished by adding mathematical formalism to the system design and implementation process. The design methodology followed on KSOS is called HDM (Hierarchical Design Methodology) and was developed at SRI International [Robinson et al. 77] in an attempt to improve the rigor of the design process. HDM is much more evolutionary than revolutionary. It takes the proven techniques for system design and implementation and adds to them the mathematical formalism needed to make precise statements about the behavior of the system. HDM also incorporates verification that successive stages in the design process are consistent with the earlier stages. Figure 1 shows HDM versus the classical software development process as defined by MIL-STD-483, -490 and -1521A. (All the services have software procurement procedures that are similar to those of MIL-STD-483, -490, but there are slight differences in terminology.)

Design Stage	HDM	"Classical"
Requirements Definition	formal mathematical model of security	broad statement of security requirements
Functional Allocation	hierarchical decomposition of system into layers of virtual machines	decomposition into functions performed by each CPCI
Functional Specification	formal mathematical specification in a non-procedural language (SPECIAL)	B5 Specifications: interfaces, input, processing and output for each function
Detailed Design	data representations, abstract programs	Structured English, flow charts, etc.
Implementation	verifiable language	language chosen for efficiency, maintainability, compatibility etc.
Requirements Compliance	formal proofs: design vs security model code vs design	manual reviews: Functional Config. Audit Physical Config. Audit

Figure 1. HDM vs Classical Design Methods

For KSOS, we have used a mixture of HDM and the classical methodology. Somewhat to our surprise the steps of HDM have been able to be incorporated into the classical methodology without great dislocation for either methodology.

KSOS will have two distinct classes of proofs made about its security properties. First, the design will be proven to be secure. This proof is independent of any implementation. It consists of proving a relatively large number (we estimate about 1000) of inequalities derived from the design. These inequalities relate the security levels of inputs for a particular function to the security levels of the outputs of that function. For a system to be secure each function must satisfy two properties:

a. the simple security property: a process may only read data at or below its security level

b. the security *-property (pronounced "star property"): a process may only write data at or above its level

The generation and proof of the theorems must be automated because there are so many of them. Fortunately, they are nearly all trivial, hence automated theorem generators and provers can be used. Presently, we have been successful at automatically generating and proving the theorems for a fragment of the design. Current efforts are underway in the area of speeding up the proof processing sufficiently to handle the anticipated number of theorems.

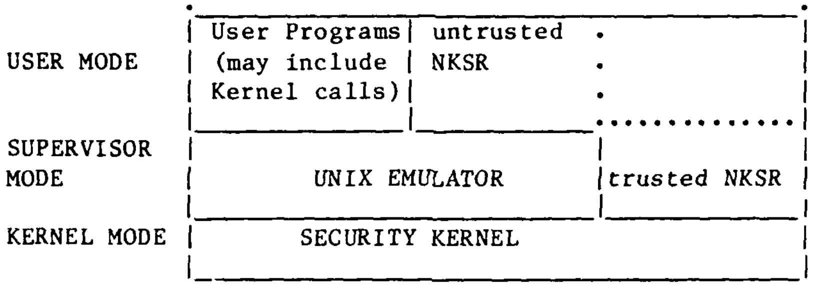
The second class of proofs about the system is that the code realizes the design. These proofs follow the methods first proposed by Hoare [Hoare 69], [Hoare 72]. They consist of showing that given a precise definition of the programming language and a set of initial assertions (the EXCEPTIONS clause from the formal specifications in SPECIAL), the program exits with the system in the "state" described by the EFFECTS clause of its specification. Full proofs of all the code in even a system as small as the KSOS Kernel are presently beyond the state of the art. We plan to do representative proofs of modules known to be both important to the overall security of the system, and whose proofs appear to be tractable.

Our experience with the design methodology has been favorable. The rigor of doing formal specifications from the very beginning of the project has been extremely valuable. Inelegant solutions to problems show up very early. A kludge in formal specifications is very, very obvious.

3. KSOS DESIGN

This discussion will be a bottom-up presentation. First, the Kernel will be discussed. The emphasis will be on the more interesting aspects of the Kernel interface, and how they could be employed for applications other than the emulation of UNIX. The UNIX Emulator will be discussed next. This will be rather brief, and will again emphasize the more novel aspects of the Emulator. The discussion of the Emulator will not present very much on UNIX per se. Finally, the Non-Kernel System Software will be discussed.

Central to the KSOS design is the notion of processes. Loosely, a process is a program in execution. In KSOS processes are comprised of two parts: a user mode portion and a supervisor mode portion. In emulating UNIX, the user mode portion is a normal UNIX application program, and the supervisor portion is the UNIX Emulator. The PDP-11/70 has three distinct memory domains: user, supervisor and kernel. The KSOS Kernel (and nothing else) resides in the kernel domain. The two process portions reside in the user and supervisor domains respectively. The user mode UNIX system calls are vectored to the UNIX Emulator which performs the internal functions and Kernel calls necessary to emulate the call. Because the UNIX Emulator is not verified, it cannot be used in any trusted services. In these cases, the user mode portion is not usrd. The program performing the trusted service resides in the supervisor domain. Figure 2 shows the relationship of the KSOS components.



(NKSR: Non-Kernel Security-Related Software)

Figure 2. Functional Components of KSOS

3.1 The KSOS Security Kernel

Viewed as an abstract machine, the Kernel's function is to create the objects of its interface (processes, process segments, files, devices, and subtypes) from the basic hardware resources of the system, and to mediate all access attempts to these objects. The Kernel is the heart of the security protection features of KSOS. The Kernel is a simple operating system that controls access to the protected objects of the system. The Kernel must allocate the sharable resources of the computer (e.g. cpu time, disk space). Due to limitations of the PDP-11/70, the Kernel manages all devices.

The Kernel enforces three distinct types of access checking. The first is the enforcement of DoD security policy. This checking is the verification of that fact that the user has the proper clearance and need-to-know to for reading the information (the "simple security property"), and that information cannot be downgraded by writing it to a file at a lower security level (the "security *-property"). The second type is the enforcement of an integrity policy described in [Biba 75]. Integrity is a mechanism for protecting system data bases, programs, etc. against modification while allowing them to be read by any process. It is formally defined to be the mathematical dual of the security model. We have found this integrity model to be overly restrictive, as its originator suspected. However, it does provide an additional, essential dimension of protection. Development of a more effective integrity model would seem to be a meaningful research topic.

The third type of access checking performed by the Kernel is discretionary access checking. Unlike the first two types of checking, the discretionary access checking is completely under the control of the user. The user may at his discretion permit or deny access by other users to the objects he owns. KSOS enforces a discretionary access policy similar to that of UNIX. For each object there are (logically) nine bits that specify read, write, and execute/search access by the owner, others in the same group as the object, and all others. We recognize that this discretionary access policy has limitations when compared to more sophisticated schemes, such as the access control lists used in Multics. However, it is simple, and requires a small fraction of the support mechanisms needed for access control lists.

The Kernel supports five different types of objects:

a. processes

b. process segments

c. files

d. devices

e. file subtypes

All Kernel objects have the same type of name called a SEID (Secure Entity IDentifier). Further, every object, regardless of its type, has a block of information associated with it that includes all the information needed by the Kernel to mediate access attempts to the object. This block is called the "type independent" information. The type independent information includes:

a. the security classification category (UNCLASSIFIED | CONFIDENTIAL | SECRET | TOP SECRET)

b. the security compartment set (e.g. NOFORN, special need-to-know compartments)

c. the integrity classification category (USER | OPERATOR | ADMINISTRATOR)

d. the integrity compartment set (presently always null)

e. the owner of the object (a user and a group)

f. the discretionary access information

Because objects, regardless of the object type, have homogeneous type independent information, access checking by the Kernel is greatly simplified. All that must be checked is that information may flow from the source to the destination. For example, if a process wishes to read a file, the source is the file and the destination is the process. In the KSOS Kernel, two functions perform all the access checking (one for security and integrity checking and one for discretionary access checking).

3.1.1 Processes

Processes are the only active agents in the KSOS design. To adequately emulate UNIX, KSOS processes must be cheap and plentiful. For example, each UNIX command is run as a separate process. Processes in KSOS will require only modest amounts of Kernel resources. Most of the Kernel data for a process will be swapped in and out with the process, reducing the amount of locked down Kernel memory space for the process tables.

Processes may possess privileges that enable them to perform functions that require reduced checking by the Kernel (e.g. changing the classification of a file) or which may require that additional checking be performed in the process (e.g. logically mounting part of the file system). The privileges that may be given to a process have been designed following the concept of "least privilege". That is, the granularity of the privileges is quite fine, and quite specific. Many service processes possess only a single privilege, and many privileges are possessed by only one process. Thus, the KSOS Kernel is designed to create encapsulated environments for critical functions. Privileges are obtained from the process image file (in UNIX the "a.out" file, in other systems this has been called a load module) from which the process was initialized. Two Kernel calls, K_invoke and K_spawn, are used for the controlled invocation of privileged software. K_invoke functions by replacing the entire process with a user-specified intermediary process. For the invocation of trusted software, this intermediary is a trusted "bootstrap" that in turn, replaces itself with the requested process image file, and sets the privileges of the process from the values in the image file. K_spawn performs the same function in a new process created as part of the K_spawn function. For both K_invoke and K_spawn, the user specifies the intermediary process image to be used. The "bootstrap" image discussed above is one choice. For other applications it may be desirable to have other possibilities for the intermediary, so that specific trusted service functions could be invoked very rapidly.

In addition to the K_spawn mechanism, new processes may be created by the K_fork call, which is similar to the UNIX fork call. K_fork creates a "clone" of the caller, a new process that is an exact copy of the caller. The only difference between the two processes (parent and child) is the return value from the K_fork call. Such a mechanism is required for the accurate emulation of the UNIX fork call.

Processes normally run at a single security level. The only exception to this is the part of the Non-Kernel System Software that changes the user's working security level. For inherently multi-level applications, the preferred design would be to create a trusted multiplex/demultiplex ("mux/demux") process which directs commands and i/o to processes running at each level needed. This would be preferable to having these per-level functions performed within one process which changes its level because such a process would be larger and more complicated than the mux/demux process. Verification of the correctness of a process becomes significantly more difficult as the process size and complexity increases. One example of this preferred architecture is the KSOS network interface. A small trusted process separates the multi-level data stream from the network into several streams. Each stream has data of only one security level in it. The mono-level streams from the processes are similarly combined by the trusted process into a single, multi-level stream.

Standard UNIX is acknowledged to be deficient in the area of Inter-Process Communication (IPC). KSOS provides significant improvements in this area. The Kernel supports both an event IPC mechanism and shared segments. The event mechanism allows one process to send a message to another process, and (optionally) to cause the receiving process to be interrupted analogously to receiving a hardware interrupt. The full set of security checks is performed for each IPC attempt. That is, information must be able to flow from the sender to the recipient, and the recipient must have permitted such information flow. Finally, a process may enable and disable the pseudo interrupt mechanism, so that it will not be interrupted during some critical operation. (Shared segment IPC is discussed below.)

3.1.2 Process Segments

A process segment is a portion of the virtual address space of a process. The process segment is not tied to the native memory management hardware of a particular machine. The KSOS process segment may be of any size from a hardware-limited lower bound up to the entire virtual address space of a process. A process may have only some of its segments actually mapped into its address space. At its creation the segment may be declared to be sharable, in which case other processes can "rendezvous" with it and map it into their address spaces. This allows for ver" high bandwidth communication between the processes. Naturally, they must estabiish a protocol that guarantees that the segment will not be corrupted through unsequenced use. The process may elect to have only some of its segments actually mapped into its address space. In particular, several segments for the same part of the address space could exist. This mechanism is used by the trusted mux/demux processes discussed above. The data segments are shared between the trusted mux/demux and the processes servicing each logical stream. The mux/demux maps in a particular segment to a well known location and puts/extracts the data for that stream into/out of the segment.

One other use for shared segments is shared text (program) segments. It is possible to have a pure text segment shared between multiple processes, thus reducing the overall memory requirements for the system. KSOS allows a segment to be locked in memory, or to be retained in the swap area for faster accessing. The designer of a KSOS-based system is offered considerable latitude in trading space for time.

3.1.3 Files and Devices

The Kernel file structure is flat and uniform. That is, there are no Kernel assumptions about the internal structure or contents of files. Directories and other higher level constructs are mechanized outside the Kernel. The UNIX Emulator creates UNIX-like directories by interpreting the contents of Kernel files. This allows a designer working directly with the Kernel to create a different type of directory structure if desired. Kernel files are accessed by blocks. There is no Kernel buffering of file i/o. Rather, the i/o is done directly into the requesting user's address space. Kernel i/o is synchronous, that is, the call does not return to the user until the i/o is completed. This is mitigated by the shared segment IPC which lets another process wait for the i/o to complete. We are currently studying the requirements for and the impact of asynchronous i/o.

Kernel devices are like a special type of file, as in UNIX. Terminals have only the lowest level echoing support in the Kernel. Higher level functions like erase/kill processing are done outside the Kernel.

KSOS supports removable file volumes. The mechanism is similar to the UNIX mount mechanism with some significant additions for protection. Because of the possibility for removing a volume, files are limited in size to one volume.

Presently the design allows for support of at least 300 Mbyte disks, with extensibility to 600 and 1200 Mbyte disks possible. These large disks may be partitioned into one or more independent extents, referred to as "mini-disks". It is possible to use a mini-disk as a device rather than as file system volume. This allows for very high performance i/o. The cost is that the process must mechanize for itself whatever structure is wishes for the raw disk device. Naturally, use of a given mini-disk as a raw device precludes its simultaneous use as a file system and vice versa.

3.1.4 Subtypes

The KSOS subtype mechanism is one of its more novel features. The subtype mechanism is designed to allow the selective encapsulation of a class of files. Each file is a member of a subtype class. "Normal" files are in the null subtype class. Files which are UNIX directories are in the "UNIX directory" subtype class. The accesses to files in a given subtype class may be restricted. For example, the subtype restriction on UNIX directories is that anyone may read a directory, but only a process whose effective user ID is the Directory Manager may write them. These subtype restrictions are in addition to the other types of access checking (security, integrity and discretionary). The access restrictions for a given subtype apply to all files of that subtype. To update a UNIX directory, the requesting process will K_spawn another, new process that executes the (privileged) Directory Manager program. This new process will perform the requested modification if possible. The mechanics of how this occurs are discussed below.

There are many other possibilities for using subtypes. For example, they could allow "peaceful coexistence" of two separate directory structures as might occur if there were two different Emulators, say one for UNIX and one for another operating system. Subtypes could also be used to control what could be done to files that mechanized the internal structure of a data base management system. Only processes that were known to correctly manipulate the structure would be allowed to change it. The subtype mechanism provides the KSOS Kernel with a significant type extension feature in that it lets the Kernel support encapsulation and control of objects without having the Kernel be cognizant of the syntax and semantics of the object.

3.1.5 Secure Terminal Interface

In a secure system it is necessary to have an "unspoofable" path to trusted services. ("Spoofing" occurs when an unprivileged user process pretends to be a privileged process. For example, a nefarious user starts a process that imitates the login sequence, and waits for an unsuspecting victim to type in his password.) In KSOS each terminal is (logically) two devices, the normal terminal device and the secure device. Only privileged Non-Kernel System Software is able to use the secure device. When the user types a reserved attention character (currently BREAK), the normal path is blocked, and the character stream is switched to the secure path. Listening on the secure path is a service process which will cause the desired secure service to be performed. Because the normal path is blocked, rather than killing off any process using it, it is possible for the user to start doing something, temporarily abandon it while requesting some secure service, and resume the activity after the secure service is completed. This is the mechanism by which the user is able to change his working security level.

3.1.6 Auditing

DoD security policy requires that certain security-related events be captured for auditing purposes. In KSOS this occurs in two ways. The Kernel captures the events it knows about and generates an IPC message to the Audit Capture process. The second mechanism is that the Non-Kernel System Software captures the event. This second case is necessary because the Kernel cannot tell that certain significant events, like a user login, have occurred. The Audit Capture process does only a minimal amount of processing and then simply places the event record into an audit log. Although it is not within the scope of the current KSOS contract, this audit log could be processed to look for suspicious (sequences of) events.

3.2 The UNIX Emulator

The UNIX Emulator is almost completely defined by its two interfaces. It must transform the system calls of the UNIX interface into sequences of Kernel calls. In the design KSOS a serious attempt was made to get a good "impedance match" between the Emulator and the Kernel, while not having the Kernel be strongly UNIX-dependent. This means that the Emulator is fairly straightforward.

The UNIX Emulator is "untrusted", that is, it has no special privileges. Thus, individual KSOS sites may modify their UNIX Emulator to provide additional functions. One of the major strengths of UNIX has been that it was easy to modify to adapt it to the needs of a particular installation. This flexibility has been retained in KSOS.

3.2.1 UNIX Directory Management

One of the major functions of the Emulator is the creation of the UNIX file system from the more primitive file system provided by the Kernel. The Emulator caches the block i/o supported by the Kernel to provide the byte stream i/o supported by the UNIX interface. The Emulator also is where UNIX directories are managed. The final design of the UNIX directory management function is the result of a long series of (occasionally heated) debates on where directories would be mechanized. Initially they were to be completely managed by the Emulator. However, this was prior to the birth of the subtype notion, and there was no way to guarantee the integrity of the directory structure. In particular, trusted software could not depend upon the directory structure. Then it was proposed to move part or all of the directory management function into the Kernel. This seemed to solve the integrity problem, but opened a new and more serious problem of making the Kernel cognizant of the structure and semantics of directory files, and thereby making the Kernel very UNIX-specific. Finally, the subtype idea was proposed. The Kernel would know that directories were "special", and would aid in the preservation of their integrity. However, the Kernel would not be aware of the internal structure or semantics of directories.

The current design has the Emulator performing all the directory interpretation functions (i.e. recursively searching for names in directories), but writing directories is only done by the Directory Manager. The Directory

Manager is a program that is K_spawn'ed into execution whenever an Emulator needs to modify a directory. It starts its life running as the user "dir_mgr" who owns the directory subtype. After getting permission for write access to directory subtyped objects, the Directory Manager reverts its identity to that of the requesting user. From there on, the the Kernel will enforce security, integrity, and discretionary access checking. Thus, the user cannot trick the Directory Manager into modifying a directory that the user cannot access. This architecture may be criticized as being too slow, since creating a new process via K_spawn is moderately time consuming. However, measurements on one of our UNIX systems in a software development environment suggest that modifications of directories is a fairly infrequent occurrence.

3.2.2 Computer Network Support

The Emulator contains the bulk of the support for the computer network interface. KSOS will "speak" Version 4 of the Transmission Control Protocol (TCP) [Postel 78b] including the Internet Datagram Layer [Postel 78a]. This protocol appears to be on its way to becoming a future standard within DoD. There are no present plans to support other protocols, in particular, the Arpanet Host to Host Protocol will not be supported at this time.

The basic structure of the KSOS network interface was discussed above. There is a Network Daemon which handles the Internet Datagram protocol, and enough of the TCP to separate the i/o stream from the network into separate streams for each connection. In each Emulator is the majority of the TCP functionality. All of the functions relating to sequence number maintenance, window maintenance, acknowledgement, and retransmission are in the Emulator. This is possible because these are per connection functions, and need not be globally managed. Although no networks presently exist that can handle multiple security levels, this architecture envisages their development and is designed to support them. To support a multi-level network, the Network Daemon would be trusted, so it could handle the multi-level stream to/from the network. The remainder of the TCP functions performed by the Emulator would be untrusted, since they are at only one level.

3.3 The Non-Kernel System Software

The purpose of this component of the KSOS system is to provide the software tools to support a KSOS system. The Non-Kernel System Software is divided into four groups:

a. Secure User Services: software that manipulates the security levels of users and files. Also included in this class are all functions that require a secure ("unspoofable") path to the service.

b. System Operation Services: software that performs continuing services for the system, such as the Network Daemon, line printer spooling and inter-user mail.

c. System Maintenance Services: software that performs occasional services primarily in the area of checking and repairing the consistency of the file system. Also included are the system generation functions. Individual KSOS sites can generate their system to suit the hardware configuration available.

d. System Administrative Services: software that aids the System Administrator in controlling the system. Our goal has been that the System Administrator need not be a computer expert to perform his functions.

The Non-Kernel System Software described is a minimally complete set. Clearly there are large numbers of additional utilities that would be desirable. It is expected that this class will be supplemented extensively as KSOS matures.

4. KSOS Application Considerations

There are two broad classes of KSOS applications, each with different considerations. The first is applications that utilize the full KSOS system, i.e. applications based upon UNIX. KSOS should appear to these applications to be only slightly different than a standard UNIX operating system. Because KSOS provides a UNIX-like interface, meaningful secure applications can be built using the existing software. UNIX is one of the best systems in existence for the creation of new products by novel combinations of existing packages, and KSOS will preserve this flexibility. Such applications can, however, be made easier in some cases via the direct use of KSOS Kernel calls. All programs may issue Kernel calls directly, but they should be careful in their use lest there be undesirable interference with the Emulator.

The second class of applications are those which do not use the UNIX Emulator, but which use either a different Emulator or which use the Kernel directly without an Emulator. The Kernel provides many features that make it an attractive operating system in its own right. It offers excellent i/o performance, a range of IPC options, and many features that ease the design of multi-level applications. Because the Kernel is "UNIX-flavored" without being heavily UNIX-dependent, it is possible to create application environments that are an amalgamation of the features provided by different operating systems.

KSOS facilitates the creation of encapsulated environments that can be used for a variety of purposes. This encapsulation allows objects to be manipulated only by software known to perform correctly. In many cases only a small part of a multi-level application actually deals with data at different security levels. By encapsulation of these functions in a small trusted process, it is possible to build multi-level applications that minimize the amount of trusted (and therefore expensive) code.

5. Summary

The KSOS project is an extremely significant one in the field of secure systems. It is moving a great deal of technology from the research community into production development. Naturally, such bold steps are not without risk. The project has blended established methods with these novel ones to minimize this risk. KSOS offers both provable security and the potential for performance close to that of a standard UNIX system. Its underlying design facilitates the creation of a wide range of applications based on the system. All major milestones have been met to date, and the project appears to be making acceptable progress towards its goals.

6. Acknowledgements

KSOS is being created by an exceptionally talented and dedicated team. It is a pleasure to acknowledge the contributions of the following people: Gerry Barksdale, Tom Berson, Ken Biba, Paul Drongowski, and Mark Gang. Ford Aerospace has had SRI International as a subcontractor in the areas of formal methodology; Richard Feiertag, Peter Neumann, Larry Robinson and Olivier Roubine have been of significant help in using and understanding the Hierarchical Design Methodology. An important acknowledgement must be made to the Government team on KSOS: Dan Edwards, Ed Burke (MITRE Corp.), Jerry Gann (formerly of MITRE Corp.), Ken Shotting, Pete Tasker (MITRE Corp.), Howie Weiss, and John Woodward (MITRE Corp.). It is a pleasure to work with such a knowledgeable and hard working team. Steve Walker now of the Office of the Secretary of Defense, but formerly of ARPA has been one of the leaders of the Government's secure systems research. His efforts made the KSOS project possible. Finally, credit must be given to Ken Thompson and Dennis Ritchie of Bell Laboratories for the creation of UNIX. We still marvel at the sophistication and elegance of their product.

7. References

This list includes several KSOS deliverable documents not referenced in the text.

[A-Specs 78] "KSOS System Specification (Type A)", WDL-TR7808 Revision 1, Ford Aerospace and Communications Corporation, Palo Alto, CA (July 1978).

[Bell and LaPadula 73] Bell, D.E. and LaPadula, L.J., "Secure Computer Systems", ESD-TR-73-278, Volume I-III, MITRE Corporation, Bedford, MA (November 1973 - June 1974).

[Biba 75] Biba, K.J., "Integrity Considerations for Secure Computer Systems", MTR-3153, MITRE Corporation, Bedford, MA (June 1975).

[Emulator 78] "KSOS UNIX Emulator Computer Program Development Specification (Type B5)", WDL-TR7933, Ford Aerospace and Communications Corporation, Palo Alto, CA (September 1978).

[Hoare 69] Hoare, C.A.R. "An Axiomatic Basis for Computer Programming", CACM, Volume 12, Number 10, pp 576-583, (October 1969).

[Hoare 72] Hoare, C.A.R. "Proof of Correctness of Data Representations", Acta Informatica, Volume 1, pp 271-281, (1972).

[Impl 78] "KSOS Implementation Plan", WDL-TR7799, Ford Aerospace and Communications Corporation, Palo Alto, CA (March 1978).

[Kampe et al. 77] Kampe, M., Kline, C., Popek, G., and Walton, E., "The UCLA Data Secure UNIX Operating System", Technical Report, University of California at Los Angeles, Los Angeles, CA (July 1977).

[Kernel 78] "KSOS Security Kernel Computer Program Development Specification (Type B5)", WDL-TR7932, Ford Aerospace and Communications Corporation, Palo Alto, CA (September 1978).

[Maint 78] "KSOS Maintenance and Support Plan", WDL-TR7810, Ford Aerospace and Communications Corporation, Palo Alto, CA (March 1978).

[MITRE 77] "Draft B5 Specifications for the MITRE Secure UNIX Prototype", Private Communication, 1977.

[NKSR 78] "KSOS Non-Kernel Security-Related Software Computer Program Development Specification (Type B5)", WDL-TR7934, Ford Aerospace and Communications Corporation, Palo Alto, CA (September 1978).

[Parnas 72] Parnas, D.L., "A Technique for Software Module Specification with Examples", CACM, Volume 15, Number 5, pp 330 - 336 (May 1972).

[Postel 78a] Postel, J.B., "Internetwork Protocol Specification", Version 4, Information Sciences Institute, University of Southern California, Marina del Rey, CA (September 1978).

[Postel 78b] Postel, J.B., "Specification of Internetwork Transmission Control Protocol - TCP Version 4", Information Sciences Institute, University of Southern California, Marina del Rey, CA (September 1978).

[Ritchie 74] Ritchie, D.M. and Thompson, K., "The UNIX Timesharing System", CACM, Volume 17, Number 5, pp 365 - 375 (May 1974).

[Robinson et al. 77] Robinson, L., Levitt, K.N., Neumann, P.G., and Saxena, A.R., "A Formal Methodology for the Design of Operating System Software," in R.T. Yeh (ed.), Current Trends in Programming Methodology, Vol. 1, Prentice-Hall (April 1977).

[Roubine and Robinson 77] Roubine,O., L.Robinson, Special Reference Manual, 3rd ed., Technical Report CSG-45, SRI International, Menlo Park, CA (January 1977).

[Verif 78] "KSOS Verification Plan", WDL-TR7809, Ford Aerospace and Communications Corporation, Palo Alto, CA (March 1978).

KSOS Executive Summary

Ford Aerospace & Communications Corporation
Western Development Laboratories
Software Technology Department
3939 Fabian Way
Palo Alto, California 94303

ABSTRACT

KSOS is the Kernelized Secure Operating System designed for DARPA. KSOS is required to be externally compatible with Bell Telephone Laboratories' UNIX*tm, to be efficient, to satisfy certain multilevel security requirements, and to be demonstrably secure. This document provides a summary of the progress obtained in Phase I of the KSOS development by Ford Aerospace and its subcontractor SRI International under contract MDA903-77-C-0333. It gives an overview of the Phase I work, including a summary of the documentation delivered under the contract. It also outlines plans for the Phase II work.

ORGANIZATION OF THIS SUMMARY

This document is organized as follows.

INTRODUCTION

The long-term goal of the KSOS effort is to develop a commercially viable computer operating system for the DEC PDP-11/70 that

- is compatible with the Bell Telephone Laboratories' UNIX*tm,
- is capable of efficiency comparable to standard UNIX*tm,
- enforces multilevel security and integrity, and
- is demonstrably secure.

In order to achieve this goal, the Phase I effort described here has designed a trusted Security Kernel and associated trusted Non-Kernel Security-Related Software, such that the trusted software:

- provides a suitable basis for KSOS;
- intrinsically supports multilevel security/integrity,
- can be used by itself to support non-UNIX*tm-based applications,and
- is able to run efficiently on a DEC PDP-11/70.

The security of the overall KSOS system must be convincingly demonstrated. This will be accomplished by formal verification of the security properties of the design (i.e., the formal specifications) and selected proofs of correspondence between the delivered code and the design. In addition, KSOS will be rigorously tested to lend added confidence in the in the system.

Although the Security Kernel is intended initially to support an Emulator providing a UNIX*tm-like user environment, the Kernel has been designed to be used by itself, or with an Emulator providing a different user environment. Typical uses of the the Kernel by itself would be dedicated secure systems such as military message processing systems, or secure network front ends.

THE BASIC DESIGN

The design of KSOS consists of a Kernel (KSOS.K) that supports multilevel security, the trusted Non-Kernel Security-Related Software (KSOS.NKSR.T) which though outside of the Kernel, is trusted to deviate internally from the multilevel security policy to provide critical system functions, an Emulator (KSOS.E) that provides compatibility with the existing UNIX*tm user interface, and the untrusted Non-Kernel Security-Related Software (KSOS.NKSR.U) providing user-level services such as secure mail and line printer spooling. As a consequence of the requirement for a convincing demonstration of KSOS security, the trusted software should be reasonably small --in order to simplify the verification effort. However, it is neither necessary nor desirable that all security-related software be a part of the Kernel, particularly because some of the security policy may vary from one application to another. The design supports various security-related functions outside of the Kernel. Any meaningful verification of security must also consider any of the Non-Kernel Security-Related Software which is trusted to violate the strict sense of multilevel security and integrity. The FACC KSOS design encourages the minimization of such trusted software, although it makes explicit the efficiency tradeoffs that arise. Note that in the design discussed here the UNIX*tm Emulator software has essentially no effect on security, and therefore does not require verification.

A slightly simplified block diagram of the design approach is given in Figure 1, showing which levels of the design depend on which others and which design levels must be trusted. A given design level in this figure is permitted to depend only on lower design levels. In principle, a particular design level may call any lower design level directly; however, in the actual implementation there will be some restrictions imposed, as noted below.

As seen in the figure, the Non-Kernel Security-Related software for KSOS is divided into two design levels, one (KSOS.NKSR.T) trusted to violate selected parts of the multilevel security model in a controllable way, the other (KSOS.NKSR.U) not requiring any trust at all. The Emulator is seen to be nontrusted. The figure shows that the trusted KSOS.NKSR.T can call upon the Kernel. It also implies that the Emulator can call upon KSOS.K and KSOS.NKSR.T. Similarly, the nontrusted KSOS.NKSR can call upon the Kernel, the trusted KSOS.NKSR.T and KSOS.E. User applications (i.e., programs or dedicated environments) may in principle use the Kernel, the Emulator, and the Non-Kernel Security-Related Software, although in the actual implementation they can be constrained, e.g., not to use KSOS.K directly. By this means, certain Kernel primitives may be restricted to use by the trusted software, and certain Non-Kernel Security-Related functions may be restricted to use by administrative officers or system daemons. On the PDP 11/70, KSOS.K will run in Kernel mode, while the trusted KSOS.NKSR and KSOS.E will run in supervisor mode. Other systems than KSOS could be built using KSOS.K, which might or might not use portions of KSOS.NKSR and KSOS.E. Implementations of KSOS or just KSOS.K on other hardware are also anticipated. In a generalized domain architecture, Figure 1 is illustrative of how the system might be partitioned into more than just three states.

It is an engineering judgment as to what should be in the Kernel, as well as to what the specific Kernel interface should be, in order best to satisfy the system requirements. The approach taken in the FACC Phase I design is expected to provide significant advantages. In this design, the Kernel provides generality suitable for the implementation of UNIX*tm and other applications, while also being modest in size and conducive to efficient implementations for these applications. This arises from the use within the Kernel of compile-time definable types (similar to the extended type mechanism in SRI's Provably Secure Operating System, PSOS). In KSOS, this mechanism is used to support multilevel secure directories, without requiring the entire directory manager to be inside the Kernel. In the case of directories, a file "subtype" is supported by the Kernel, while the directory manager is a part of KSOS.NKSR.T. This allows the integrity of the directories to be improved while continuing to allow the Emulator to be untrusted.

The methodology employed throughout facilitates verification that the entire system satisfies the desired multilevel security properties. This verification is composed of two parts. First, that the design is consistent with the formal requirements, and second that the implementation is completely consistent with the design. As a result of the latter verification, the security of the implementation can be effectively demonstrated. Moreover, note that much more is thereby verified since the consistency proofs of the implementation guarantee not just secure operation but also correct operation, assuming the specifications are correct. That is, the demonstration that programs are consistent with their formal specifications guarantees that the implementation does what is specified, no more, and no less. It should be

```
                           |
......                     V  K+T+E+U+A                 .......
                |-----------------------------------|
                |       UNIX*tm Applications        |
                |            Untrusted             A|
                |-----------------------------------|
User                       |
mode                       V  K+T+E+U
                |-----------------------------------|
                |Non-Kernel Security-Related Software|
                |       Untrusted portion           |
                |         KSOS.NKSR.U              U|
                |-----------------------------------|
.................          |
                           V  K+T+E
                |-----------------------------------|        ^
                |        UNIX*tm Emulator           |        |
                |           Untrusted               |       Not
                |            KSOS.E                E|     Trusted
                |-----------------------------------|
Supervisor                 |                    .....................
mode                       V  K+T
                |-----------------------------------|     Trusted
                |Non-Kernel Security-Related Software|        |
                |        Trusted portion            |        V
                |          KSOS.NKSR.T             T|
                |-----------------------------------|
.................          |
                           V  K
                |-----------------------------------|
                |         Security Kernel           |
Kernel          |            trusted                |
mode            |            KSOS.K                K|
                |-----------------------------------|
......                                                  .......
```

Figure 1
Block Diagram of KSOS Components

Note: K,T,E,U,A denote the functions provided by the five levels in upward order, respectively. The interfaces potentially visible at each level are cumulative upwards, e.g., as indicated by K+T+E+U+A. In actual implementation there may be restrictions on function visibility.

remarked that this two step verification, first of the design and then of the implementation, may reduce the overall verification effort. It also allows strong statements to be made about the system design whether or not full code proofs are undertaken.

The work of this contract has taken a strong systems viewpoint toward the overall development of the Security Kernel, the Non-Kernel Security-Related software, and the UNIX*tm Emulator. This viewpoint is focused around the use of a formal methodology for system design, implementation, and verification that has been developed at SRI International, and used previously on various system designs. The methodology is called the Hierarchical Development Methodology (HDM). Its use permits a wide collection of needs arising throughout the development and subsequent use of the Security Kernel and its surrounding KSOS software to be carefully addressed or anticipated. As a consequence, the resulting KSOS design provides:

- multilevel security;
- provable security;
- high reliability and availability;
- high performance (operational efficiency) of both the Kernel and the UNIX*tm Emulator;
- flexibility of the Kernel design to be readily applicable to other hardware bases besides the PDP-11/70 (e.g., to the Honeywell SCOMP);
- generality of the Kernel design to be applicable to other security-relevant applications instead of or in addition to KSOS, e.g., a dedicated message processing system;
- controllability of the maintenance and evolution of the Kernel and Non-Kernel Security-Related software;
- ease of maintenance, evolution, and particularization to installation needs of the Emulator software, without adverse impact on the overall system security.
- ease of reverification following changes to the trusted portions of the system (KSOS.K and KSOS.NKSR.T).

It should be noted that the goal of provable security has significant implications that would affect any development process, with respect to the design, the choice of specification language, the choice of the programming language, and the choice of the verification methodology. However, these are all addressed by HDM and by the approach taken here.

THE HIERARCHICAL DEVELOPMENT METHODOLOGY, HDM

The formal methodology used in Phase I and proposed for use in the Phase II development of the KSOS system is summarized below.

- An overall systems viewpoint is maintained throughout.
- A unified methodology is used for design, implementation, and verification. This greatly increases the understandability of the design, the ease of implementation, and the verifiability of the resulting system. It includes the use of a formal specification language called SPECIAL (A SPECIfication and Assertion Language).
- The methodology encourages a hierarchically decomposed design, which itself has strong implications on initialization, shutdown, recovery from hardware and software errors, maintenance, and verification.
- A programming language is to be used that is well suited to both system programming and to eventual program verification.
- Verification is separated into two distinct stages, the first showing the correspondence between the formal specifications of the design and the formal requirements for multilevel security, the second showing the consistency of the programs with their specifications. The combination of these stages assures that the implementation completely satisfies the multilevel security requirements. This approach increases the understandability of the proofs, and also simplifies them.
- Advanced but well-debugged development tools supporting HDM have been used and will be used wherever appropriate. Existing tools used in Phase I include checkers for the hierarchical structure, the specifications, and the mappings between the state representations at different levels.

An existing theorem prover and simplifier are expected to be used in Phase II to provide verification tools supporting proofs of correspondence between specifications and the multilevel security model. Related tools -- some existing and some under development -- may be used to provide illustrative proofs of program correctness, as appropriate.

The methodology attempts to unify the entire development process. It decouples design and implementation into distinguishable stages, providing a formal definition of the design and a formal basis for implementation and proof. This approach considers the entire development process in a formal way and permits formal proofs at each stage in the process. Even in the absence of proofs, this approach seems tc greatly increase the understandability and precision with which a design can be expressed, and the ability to evaluate the reasonableness of such a design with respect to stated desired properties of the system. The methodology has considerable utility throughout the development of KSOS, in Phase I, in Phase II, and in any additional efforts to provide proofs of implementation correctness. It also makes a positive contribution to various further related tasks, such as verification of the consistency of any subsequent changes affecting security, as well as implemention of the design on other hardware and verification of the resulting system. In the latter case, specifications for most of the Kernel (except for the machine and device-dependent levels) could remain largely intact, and the specifications for KSOS.E and KSOS.NKSR.T could remain unchanged. Thus the demonstration of the security of the design can carry over directly to the new implementation. The verification of consistency between code and specifications might also carry over in part, depending on the programming language used.

SECURITY

The desired multilevel security requirements demand that information at a particular security level may not move downward to a lower security level. Because of the syntax of SPECIAL, the proofs that these requirements (formally stated) are actually satisfied by the specifications follow largely from simple (i.e., mostly syntactic) checks on the specifications. Following such proofs, any implementation consistent with the specifications would itself satisfy the security requirements. That a design proved to be secure is itself correctly implemented then follows completely from proofs of the consistency of the specifications with their implementing programs and hardware. (The dependence on correct hardware is made quite explicit by this approach.) It is of course also desirable to demonstrate that the specifications --even if proved to be secure-- actually describe the desired effects. This task is aided by the understandability of the specifications, and by testing of the resulting implementation. For example, the specifications for the top-level (user-interface) can be compared with the behavior of existing UNIX*tm in the case of the Emulator. The resulting system can be compared with exisiting UNIX*tm by running programs and applications environments on both systems.

The design for the Kernel permits all of the Kernel primitives to satisfy the desired security properties completely under normal usage by users. A few relaxations of this strict behavior are necessary to support the trusted Non-Kernel Security-Related software, and are confined to the KSOS.NKSR.T by the controlled distribution of minimal privilege. These isolated relaxations can be shown to satisfy a specific subset of the security properties, in a completely controllable way, and to be masked completely by the trusted Non-Kernel Security-Related software.

THE ROLE OF SPECIFICATIONS

Formal specifications by themselves provide a significant advance in the state of the art of software system development. They provide a concise and precise functional statement of exactly what any external or internal interface is expected to do. They enforce abstraction on the design that consequently simplifies implementation, debugging, system integration, and maintenance. They greatly enhance the understandability of a design. They provide a forum for discussion of design issues. Their understandability encourages the manual discovery of design errors. They also make possible the intuitive verification of certain desired properties that the design should satisfy.

THE ROLE OF THE PROGRAMMING LANGUAGE

It is desired that the programming language used for the Kernel and the Non-Kernel Security-Related software have certain strong properties. (The Emulator may also take advantage of this language.) The desired properties include such things as

* adequate compiler support for generating efficient code,
* suitable constructs for control and data abstraction,
* type safety,
* ability to support multiprogramming, and
* ability to handle machine-dependency when necessary.

Some of these desired properties (notably type safety and support of abstraction) contribute significantly to the verifiability of the resulting code. They also contribute to the avoidance of many characteristic security flaws. At the moment, Euclid appears to be highly appropriate, with an extended Modula as an alternate choice. (It appears that some of the competitive DoD/1 languages would be appropriate, if adequate support were available.)

THE ROLE OF VERIFICATION

As noted above, specifications support proofs of specification properties, and also facilitate proofs of program consistency with the specifications. The ability to state and prove properties about a design (as represented by a set of specifications) -- before that design is ever implemented -- will have a significant impact on the system development. Nevertheless, no system can justifiably be thought to be secure unless appropriate properties of its implementation can also be proved. On the basis of the work to date, proving that the specifications for the KSOS design satisfy the required multilevel security properties can be straightforward and accomplished largely by automated tools -- many of which have already been developed at SRI. In addition, although more complex than such design proofs, proving the consistency of implementation with respect to the specifications is now becoming a realistic task, especially with the emergence of recent theoretical advances and the prospect of suitable on-line tools. Furthermore, the expected use of a language like Euclid or extended Modula would very helpful. In addition, the proposed use of review and testing is expected to increase the confidence in the implementation.

THE ROLE OF ON-LINE TOOLS

The role of computer tools is indicated above, with respect to the syntactic checking of specifications, the verification of the security of the design, and the eventual verification of the consistency of programs with the specifications. Experience in attempting to develop secure systems in the past indicates that an enormous amount of mind-numbing effort would be required under conventional approaches, and even then there is considerable doubt as to whether security flaws still remain. The approach outlined here, with its judicious use of on-line tools that support the Hierarchical Development Methodology, is expected to result in considerably more confidence in the security of the resulting system than is possible with conventional, largely manual approaches. Further, the automated approach promises to be far more cost-effective. For example, during the exercise of writing of formal specifications for UNIX*tm, various previously unknown flaws in that system were detected. In the writing of formal specifications for the KSOS Kernel, various minor flaws were detected by the hierarchical interface checker and the specification analyzer. These flaws, many of which might give rise to insecurity in the implementation, have been detected and removed during this early stage of design. This is particularly valuable for various minor typographical errors in the specifications which otherwise might result much later in significant flaws in the resulting system. In addition, because of the structure and constraints of the methodology, flaws in the implementation of even a correct design may also often be detected by the implementation tools, e.g., the compiler and simple consistency checks.

THE KERNEL

The Security Kernel (KSOS.K) is structured into a hierarchically ordered set of modules, each of which depends (for its implementation and for its correctness) solely on lower-level modules. The set of accessible Kernel calls has been chosen to be powerful and efficient for the implementation of KSOS, but general enough for the implementation of other applications (e.g., dedicated). These Kernel calls support (among other things) the creation and deletion of files and processes, the reading and writing of files, interprocess communication, and the protected invocation of trusted software.

The Kernel has a "UNIX-flavor" to it. It was designed with the actual implementation of the lower levels of UNIX*tm in mind. This, of course, does not mean that the Kernel is suitable only for creation of UNIX*tm user environments. Significant efforts have been made to make the Kernel both machine independent and UNIX*tm independent. The Kernel design incorporates many of the concepts from the existing prototype "Secure UNIX*tm" implementations. Its main departure from the prototypes is that the FACC design does not employ virtual memory. This decision was reached because existing UNIX*tm software has very large "working sets" that minimize the value of a virtual memory architecture. Also motivating against a virtual memory architecture are the long delays associated with process environment switches on a PDP-11/70. Satisfying page faults, even if the page is in core could significantly degrade system performance.

The Kernel internally supports objects of program-definable types and capability addressing. These are intended for use within the Kernel for creating Kernel-supported objects such as multilevel secure directories without requiring any of the directory mechanism to reside within the Kernel

-- the directory manager is in KSOS.NKSR.T. An overview of a proposed design decomposition of the Kernel follows, from highest level of abstraction to the lowest.

* Kernel calls
* process operators
* interprocess communication
* file capabilities
* file subtypes
* process segments
* process states
* mountable file systems
* file contents
* file states
* multilevel security
* privilege control
* device-independent functions
* type-independent information
* secure entity names

THE TRUSTED NON-KERNEL SECURITY-RELATED SOFTWARE

Only part of the Non-Kernel Security-Related Software must be trusted (and hence ultimately verified). Although most of the Non-Kernel Security-Related functions must contain a small amount of trusted code, most of the code supporting these functions need not be trusted. A spectrum of design decisions can be made either distributing or centralizing the trusted portion of each function. The FACC design permits the portion which must be trusted to be kept small. The Non-Kernel Security-Related Software as a whole supports the following functions.

* system startup and shutdown
* login and logout
* password changer
* user security-level changer
* file security-level changer
* virtual terminal handler
* mount and unmount
* line-printer daemon
* file system maintenance, dump/restore
* system administration

As noted below, the spooler and the mailer are examples of security-related programs that do not need to be trusted, because of the constraints imposed by the Kernel and the trusted Non-Kernel Security-Related software. The nontrusted functions need not be verified. Further simplifying the verification effort of the trusted portions is the fact that they are composed of autonomous modules which can be verified independently.

THE EMULATOR

The KSOS Emulator interface supports the UNIX*tm calls, and implements them in terms of the KSOS Kernel. It is protected from the user, and the Kernel is protected from it. In general, it calls the Kernel directly rather than going through the trusted Non-Kernel Security-Related software, except for certain directory operations. In essence, the Emulator does whatever it has to in order to provide compatibility with the desired UNIX*tm calls.

However, certain features of UNIX*tm have been removed from the user interface to KSOS, in the interests of providing a secure system. Most notable among these is the "superuser" facility. Also, the checks on certain user functions have been strengthened.

The Emulator contains the bulk of the support for the interface to the computer network. Only the multiplexing and demultiplexing of the data streams to and from the network are trusted. The flow control and data stream integrity functions of the network are untrusted and are supported on a per-process basis by the Emulator. This architecture is extremely attractive for a number of reasons. First the size of the trusted software is reduced to a minimum. Second, the flow control is truly end-to-end. Third, overall structure requires minimal Kernel support. Finally, the basic architecture can be easily adapted to support other networks protocols.

THE NONTRUSTED NON-KERNEL SECURITY-RELATED SOFTWARE

As noted above, many of the Non-Kernel Security-Related functions require some trusted code, although most of the code for the implementation of these functions need not be trusted. In addition, the spooler and the mail facility -- although in principle security related -- can operate entirely as untrusted programs. The design thus allows great flexibility in its implementation. It is also possible to easily extend the functions provided by the Non-Kernel Security-Related software because they are not hard coded into the Kernel.

THE WORK PROPOSED FOR PHASE II

The aim of the proposed Phase II work is to develop an effective implemention of the design Phase I KSOS design, to demonstrate that this design completely satisfies the desired properties of multilevel security, and to demonstrate the essential correctness of the implementation by illustrative rather than exhaustive means. On the basis of the design that has emerged from Phase I, and the structured methodological approach being used throughout the development, there is reasonable evidence that this aim can be accomplished in a timely and cost-effective way. The proposed work for Phase II will also provide detailed illustrations of how the implementation can be demonstrated to be correct, that is, proven consistent with its specifications.

PRELIMINARY EVALUATION

The approach used here affords various significant advantages over previous competing approaches, but avoids incurring many of the risks typically associated with high-technology attempts to advance the state-of-the-art. Considerable success has already resulted from the use of this approach, and such success is justifiably expected to continue.

From a systems viewpoint, the work described here is novel in many respects. These include the following.

* KSOS will be the first full use of the formal methodology (HDM) for a complete system development. However, HDM has been well tested in the design stage of several previous projects.
* The HDM methodology can accommodate the verification of a larger amount of Kernel and other trusted software than can other approaches. This is due to two orthogonal decompositions: the decomposition of the

verification process into stages (e.g., specification-to-model proofs, followed by code consistency proofs) and the decomposition of the design into hierarchical levels of abstraction. These both simplify the verification effort significantly. The automated tools offer a manyfold further reduction in effort. In addition, the approach is directly applicable to the verification of the security of the Non-Kernel Security-Related software.

- KSOS is likely to involve the first use in the development of a production-quality computer system of a modern programming language (Euclid, or possibly Modula) highly appropriate for such an effort. Note that each of these languages is a conservatively designed variant of an existing well-established programming language (Pascal).
- This will be the first implementation of a production system that includes a Security Kernel designed to be provably secure, and implemented using a programming language suitable for such verification.
- The design takes advantage of several innovative operating system concepts, e.g., using objects of extended type (here called file subtypes) within the Kernel. The use of Kernel-supported types is expected to produce significant advantages in flexibility and generality.
- Because of these innovations, it should be stressed that the risks are minimal. The experience to date is very promising. For example, the time required for FACC to master the methodology was shorter than expected. The approach is significantly aided by well-used supporting tools. The task of formally verifying that the specifications for the KSOS design satisfy the multilevel security requirements seems reasonable. The task of producing an efficient and secure implementation from the existing Phase I design appears to be straightforward. The task of demonstrating that the implementation is correct ultimately requires formal proofs that the programs are consistent with the specifications. While complete proofs are not proposed, it is expected that a combination of illustrative proofs will demonstrate the feasibility of carrying out complete proofs in the future.

The design takes advantage of the strengths of both of its prototype precursors, namely the UCLA Data Secure UNIX*tm and the MITRE Secure UNIX*tm, although the present approach has numerous advantages over those prototypes, as follows.

Re UCLA: The FACC design carefully considers efficiency and flexibility in advance. (Note that the use of capabilities within the Kernel is also found in the UCLA Kernel.) The use of formal specifications with a proof methodology tied to those specifications permits proofs of the intrinsic security of the design, based on the specifications, independent of subsequent implementation and verification of implementation correctness. The FACC choice of programming language seems to be better suited for implementation and for eventual program verification than UCLA Pascal.

Re MITRE: The SRI formal methodology for specification and proofs of specification properties is similar to that used by MITRE; however, the concept of hierarchy, the specification language, the program proof methodology and the tools for automatic specification checking and program verification are more advanced than MITRE's.

The FACC KSOS design does differ from the protoypes in that it does not use virtual memory. As discussed above this choice was motivated by performance considerations, and analysis and experimentation with virtual memory UNIX*tm systems.

GUIDE TO DOCUMENTATION

The following documents are included in the documentation of the KSOS Phase I effort.

KSOS System Specification (Type A)
KSOS Computer Program Development Specifications (Type B5)
KSOS Verification Plan
KSOS Implementation Plan
KSOS Maintenance and Support Plan

KSOS SYSTEM SPECIFICATIONS (TYPE A)

The System Specification (Type A) establishes the requirements for the KSOS system with respect to performance, design, development, and test. Deviations from the behavior of the existing UNIX*tm user inerface are explicitly cited.

KSOS COMPUTER PROGRAM DEVELOPMENT SPECIFICATIONS (TYPE B5)

The Program Development Specifications (Type B5) provide the detailed design of the Kernel, the Non-Kernel Security-Related software, and the UNIX*tm Emulator, with one document for each. The interface presented by the Kernel is given in detail. A draft version of formal specifications (written in SPECIAL) for the externally visible functions and many of the internal functions of the Kernel is included as an appendix to the Kernel B5 specs. These are not required in final form until Phase II, but are included at this time as illustrative of the approach, and demonstrative of the depth of consideration given to the design. Preliminary formal specifications for the existing UNIX*tm system exist and have been distributed previously, although they are not required at this time. The process of generating these latter specifications was very helpful in defining what KSOS should actually appear to do, and was also valuable in ferreting out several hitherto unknown bugs in UNIX*tm.

VERIFICATION PLAN

The Verification Plan provides the precise model for multilevel security that the Security Kernel is expected to satisfy. It also shows how the formal specifications for KSOS can be formally proven to be consistent with the formal model for multilevel security. In addition, it discusses the choice of programming language to be used in the Phase II implementation, the process of verifiying that programs are consistent with the formal specifications (in Phase II and beyond)·, and the tools that would be used to support the verification effort associated with the Phase II development effort.

IMPLEMENTATION PLAN

The Implementation Plan discusses programming techniques, implementation tools, testing, external configuration management, and the assurance of integrity and performance of the implementation. FACC plans to utilize its on-going work in development-support systems based on UNIX*tm to aid in the creation of KSOS. The plan emphasizes tools that are well matched to the scope and nature of the KSOS effort.

MAINTENANCE AND SUPPORT PLAN

The Maintenance and Support Plan discusses what will be required in order to test, maintain, and modify the KSOS software. The long term maintenance of KSOS is viewed as an extention to the procedures for configuation management and trouble reporting that will be routinely used during the development phase. Thus, the mechanisms will be well established and thoroughly "debugged" prior to the maintenance phase. Also discussed in this document is the mechanism for system generation of KSOS at the individual user sites. The procedures are intended to allow a security officer (or other similarly computer-naive users) to generate a KSOS system, and to be assured of its security and integrity properties.

SOFTWARE ENGINEERING: TOOLS & METHODS I

Derek S. Morris
CENTACS

SOFTWARE ENGINEERING TOOLS & METHODS I

SESSION CHAIRPERSON: Derek S. Morris

CENTACS

SESSION SUMMARY

This session explored the impact that the Department of Defense Common Programming Language will have on program development methodology and its unifying effect on tool requirements. The session began with a discussion of the history of the common language project and a synopsis of the language itself. A major portion of this language project has been the development of a set of technical requirements on programming languages for these kinds of applications. At the present time two competitive language designs are underway to satisfy these requirements, one of which will be selected in the Spring.

It is recognized that various potential users of the language are concerned that the resulting language may not be applicable to their particular area and are further concerned that they may ultimately be forced via DOD policy to use the language where they feel it does not apply. It was the objective of the second paper to attempt to alleviate this concern by discussing the technical capabilities of the language relative to some of the major technical problems that arise in telecommunication system software designs. As an example, a 50-line message switch such that would be used in the worldwide defense communication system designed and coded in terms of the features that the common language will possess.

Lastly, the discussion focused on the support tools and program development environment needed by the language. Support software tools are typically a collection of independently designed programs which support no specific higher order language and which provide a non-uniform and often unfriendly user interface. The third paper described an integrated system of cooperating tools which supports DOD common language program development in a friendly and powerful environment.

The Department of Defense Common Programming Language Project

Serafino Amoroso

CENTACS

The Department of Defense is attemping to improve the quality and reduce the cost of developing and maintaining the software for its many computer-controlled systems. A major portion of this effort has been the development of a set of technical requirements on programming languages for these kinds of applications. At the present time two competitive language designs are underway to satisfy these requirements, one of which will be selected in the Spring. This paper reviewed the history of the project and gives a preview of what the new language may look like.

THE DEPARTMENT OF DEFENSE
COMMON PROGRAMMING LANGUAGE PROJECT

Dr. S. Amoroso

US Army Communications Research
and Development Command
Center for Tactical Computer Systems
Software Engineering Division
Fort Monmouth, New Jersey

Studies were conducted for the Department of Defense in 1973 and 1974 concerning the costs of software being procured by the DoD (e.g., [1]), costs that were very high and increasing at an alarming rate. These studies showed that the greatest software problems were associated with what are now referred to as "embedded computer systems". The total costs for the software for such systems exceeds half of the total of all DoD software costs. Examples of non-embedded computer system software would include management information systems which account for approximately 19% of the total software costs, and the software for scientific computing which accounts for only 5% of these costs.

These concerns about software costs resulted in a major effort directed by the highest levels in DoD to see what could be done about reducing these costs and also improving the quality of the software being procured, which was another serious problem. Since the greatest potential benefits could be obtained by concentrating on the problems with embedded computer software, and since it was determined that a serious problem existed in programming language usage in this application area, a DoD-directed tri-service working group was established. We will refer to this group here as the HOLWG (the high-level language working group). Its mission was in effect to set some standards for language usage in this application area and to make available the most appropriate languages and supporting tools. LTC William Whitaker has represented DoD and chaired the group from its inception, and Dr. David Fisher from the Institute for Defense Analysis has been chief technical consultant also from the very beginning.

Before continuing with this history we should take a closer look at what we have been calling "embedded computer software". This would include the software for computers that are dedicated to and a part of larger systems such as tactical systems, shipboard systems, aircraft control systems, and communication systems. We would also include under this heading the software necessary to design, develop, and maintain such software systems.

Embedded computer software is often quite large, long lived, and subject to a great deal of modification and improvements during its lifetime. It is not unusual for annual revisions to be of the same magnitude as the original development. Personnel turnover for those associated with the maintenance of such systems is rapid, typically two years.

The reliability of such software is often crucial involving the safety not only of costly equipment, but often of human life. Hence, it is often essential that such systems continue to operate in the presence of faults in the input information, in operator procedures, as well as in the software itself. Such software must interface a great diversity of input-output devices including control signals, analogue devices, and sensor monitorings. Such software must support the treatment of logically concurrent activity and must be capable of responding to critical timing problems, i.e., service interrupts often have to be handled within critical time periods to avoid disasterous consequences.

All of this implies the following major requirements for programming languages for embedded computer applications:

- A capability of aiding the process of producing reliable, robust software, i.e., software that is correct and capable of operating in the face of various kinds of adverse conditions.

- A capability of aiding in the development of modifiable software, i.e., software capable of being extended, changed, and improved in a controlled rational manner.

- A capability of handling logically concurrent processes.

- A capability of interfacing a wide variety of low-level I/O devices.

- A capability of interacting with real-time clocks to control devices in response to time constraints.

Although the use of high-level languages for embedded computer software is widespread, most such software is still developed (and hence maintained) in machine-level languages. There is little question that the readability of such code is very poor, and making modifications to machine-level code is as frustrating and error-prone as any imaginable endeavor. Obviously, the unnecessary use of machine-level language for embedded computer software is a factor contributing to the DoD software problems.

Even when embedded computer software is programmed in higher-level languages, serious problems still exist. At the time the HOLWG was formed there was no control on usage of languages for these applications. It was not rare for a major project to begin by designing its own high-level language, or at least modifying an existing language to an extent that in effect a new language was created. These would be no small undertakings. Systems programming capabilities would also be needed to develop and maintain support software including translators, software development tools, testing aids, as well as host operating systems and special purpose executives. Such costly duplication would obviously divert from the effort of developing the application system itself. The cost of developing such language systems almost always meant that only the most primitive programming aids could be afforded. Finally, having systems tied to such special purpose language systems would mean unnecessary further ties of maintenance to the original developer.

The large number of duplications of these essentially similar language systems, across the Services and even inside any one Service, obviously has been another contributing factor to the high cost to DoD for software. We should emphasize that the costs associated with the development of the supporting software for these duplications is especially severe.

Additional problems caused by using different languages for essentially similar tasks are: the research problems associated with embedded computer software are scattered and diluted due to a resulting lack of focus. Related to this is the lack of communication and technology transfer among software practitioners working with different language systems.

This was the situation that existed at the time the HOLWG was formed in January, 1975. One of the first early definite steps taken to place some controls on this free usage of programming languages in the DoD was the issuance of DoD Directive 5000.31. This is a small list of "approved" high level languages that can be used to develop new embedded computer application programs without further approval. The use of any other language not on this list must be justified, and such justification would have to show benefits over the life-cycle of the project and not just developmental savings. This justification was intended to apply to the use of machine-level languages as well (in fact especially so). The directive has been in effect less than two years with little adverse reaction.

Besides placing some meaningful controls on the free use of programming languages in the DoD the mission of the HOLWG includes making available for DoD software development the most appropriate languages and supporting tools possible.

In order to appreciate the main activity of the HOLWG, which has been developing for almost four years now, and which will be discussed below, the following must be understood:

- Almost all the high-level languages currently being used for embedded computer applications represent the programming language technology of at least ten years ago.

- The entire history of high-level languages spans only some twenty years.

- Programming language design is a very active and fruitful topic with many important developments over the past ten years.

In the spring of 1975, the HOLWG issued a strawman on the requirements for programming languages for embedded computer applications. The document resulted in a large number of worthwhile comments which gave rise to a major rewrite of these requirements dated August, 1975 and called informally the Woodenman document. This document was widely reviewed not only in this country but in many countries in Europe as well. The comments of every potential user, of industry, and of the research community were activily solicited. Again, a very rich response of constructive criticism was received. The next major rewrite called Tinman issued in January, 1976 was at the time believed to be the final requirements.

In 1976, a major effort was sponsored by the HOLWG with cooperation by the three Services to compare the Tinman set of language requirements against a large number of existing standard programming languages (in fact twenty-three languages!). Six US industrial firms plus many voluntary teams contributed to this study. The results were surprisingly free of disagreement, and can be summarized as follows:

- None of the existing languages satisfies the requirements so well that it could be adopted with minor changes as a common programming language.

- It was the consensus of those that worked on this effort that it is currently possible to produce a single language that would meet essentially all the requirements.

Based on the technical discussion from this effort plus the results of a workshop on implementation issues (Cornel University, October, 1976), another version of the requirements was necessary. This was the Ironman document issued in January, 1977 [2].

Based on the Ironman set of requirements, language designs were solicited. A large number of proposals were received, some from European groups. Just about all of the leading names in programming language design were associated with one or another of these proposals. After a detailed evaluation, four of them were supported by Service funds coordinated and administered by the Defense Advanced Research Projects Agency (DARPA). In February, 1978, four preliminary language designs were received (from SofTech, Intermetrics, Stanford Research Institute, and Cii Honeywell Bull). An extensive two month evaluation of these designs involving over sixty technical teams, many from outside the US, resulted in the Intermetrics design and the Cii Honeywell Bull designs given another ten months of support to complete their designs. This is the current state of affairs with the completed designs due early in 1979.

Besides this language design work, the HOLWG is also developing the requirements for supporting tools for the language, for compilers, for controlling organizations, and for implementation validation procedures. The details of all of this must be delayed for another time.

We would like now to complete this paper with a brief preview of some of the technical characteristics of the emerging new language.

An Encapsulation Facility:

The idea of programming languages having facilities for partitioning programs into intercommunicating units, exporting precisely what is needed by other units and hiding all detail irrelevant to other units, began with the introduction of the CLASS concept in the SIMULA language. This concept has been developing in recent years and is closely related to the topic of abstract data types, a currently active research area in language design. Such a program structuring facility is expected to have major application in program maintenance and reliability.

The DoD language is expected to have an encapsulation facility very much like that of the "module" found in the language MODULA [3]. The language allows the "encapsulation" of program units in that certain data structures and operation designed explicitly for these structures are explicitly made available for use outside the encapsulation. All other parts of the encapsulation are protected by built-in compile-time, language defined restrictions. Hence, modification to non-exported parts of an encapsulation can have no effects on the rest of the system (besides certain possible efficiency effects). Such encapsulations are processed completely at compile-time and have no run-time associated costs.

The Treatment of Concurrent Processes:

The introduction of facilities into programming languages to directly present to the user the notion of concurrently acting processes was introduced in the mid 1960's. This again is one of the most active research areas in computer science. The topic of synchronizing concurrent processes competing for shared resources has developed from semaphores to critical regions to monitors to a new technique published less than two weeks ago. It is not clear what will eventually be used in the DoD language, but safe, well-understood features are essential.

Those of us who have had some experience using languages with such facilities find them extremely useful. The clarity and naturalness that follows from such facilities are sure to contribute to readability and maintainability.

The Treatment of Exception Conditions:

As we mentioned earlier, many defense software-driven systems must continue to operate even in the face of abnormal conditions such as damage to some part of the hardware or faulty inputs to the system. These problems have been handled to one degree or another in most languages (e.g., FORTRAN and ALGOL) and some languages, even as far back as PL/I have designed into them facilities for dealing with these problems in a direct way. The topic is known today under the name "exception handling", and is another current research area. A great deal has been learned about this topic since the early attempt by the PL/I design. Again, the DoD language is expected to have a conservative, safe facility that should be an improvement over what is available.

Interfacing with Low-Level I/O Devices:

All of the preliminary language designs delivered in February were disappointing on their interfacing facilities for low-level I/O devices. The hope was that there would be an improvement over the usual treatment which is essentially machine-language descent. A recent study seems to promise hope for the future [4]. The most we can probably expect in the next years is an inspired use of the language's library facilities for this I/O requirement.

Summary:

The HOLWG has made a serious effort to solicit the views and advice of almost every qualified person from the user community, the Services, industry, and the research community. The development of the requirements underwent continuous scrutiny and no constructive criticism was ignored. The DoD language will obviously have its problems. Any new language will. But it is our firm belief that the language that finally arrives next year will be a useful language, significantly more suited to helping solve the software problems of DoD than the programming languages currently available.

References

[1] D. A. Fisher, "Automatic Data Processing Cost in the Defense Department," Institute for Defense Analysis, Paper P-1046, AD-A004841, October 1974.

[2] Department of Defense Requirements for High-Order Computer Programming Languages - Revised IRONMAN, July 77. (Now Steelman, June 78).

[3] Wirth, N., Modula: A Language for Modular Multiprogramming. Software Practice and Experience 7 (1977), 3-35.

[4] Perry, D., High-Level Language Features for Handling I/O Devices in Real-Time Systems. PhD Dissertation, Stevens Institute of Technology, 1978.

An Experimental Application of the DOD Common Language to a Telecommunications System Design

Derek S. Morris

CENTACS

It is recognized that the Department of Defense Common Language may have wide spread influence as it embraces participation from the three armed services, NATO and the European community. This recognition has lead to leading international computer language designers contributing to the design effort. It is also recognized that various potential users of the language are concerned that the resulting language may not be applicable to their particular area and are further concerned that they may ultimately be forced via DOD policy to use the language where they feel it does not apply.

It was the objective of this paper to attempt to alleviate this concern within the telecommunications area by discussing the technical capabilities of the language relative to some of the major technical problems that arise in telecommunication system software designs. As an example, we take a 50-line message switch such that would be used in the worldwide defense communication system and design and code it in terms of the features that the common language will possess. In order to keep the discussion within a palatable scope, we concentrated on a particular important issue and mentioned the others in passing. In that way we were able to treat the entire design process from a functional description right down to code in the language and include a discussion of the semantics of the resulting code.

AD-A104 252 INTERNATIONAL BUSINESS SERVICES INC WASHINGTON DC F/G 9/2
US ARMY SOFTWARE SYMPOSIUM (2ND) HELD AT WILLIAMSBURG, VIRGINIA--ETC(U)
1978 S M TAYLOR DAAK70-78-D-0030

UNCLASSIFIED NL

3 OF 10
A
A104252

AN EXPERIMENTAL
APPLICATION OF THE DOD COMMON LANGUAGE TO
A TELECOMMUNICATIONS SYSTEM DESIGN

Derek S. Morris

US Army Communications Research
and Development Command
Center for Tactical Computer Systems
Software Engineering Division
Fort Monmouth, New Jersey

1. Background of the Common Language Effort

The Department of Defense spends about $3.5 - $4 billion annually on computer software. This includes the design, development and maintenance (post-deployment correction and enhancement) of such software. Only a small fraction of this effort is involved with what is considered the mainstream of Automatic Data Processing; e.g. accounting, inventory, payro'ling, and financial management. These functions have their exact analogy in the commercial sector and share a common technology, both hardware and software. A much larger fraction of the DOD's computer investment is in computer resources which are embedded in, and provided as part of, major weapons systems, communication systems, command and control systems, etc. Major factors contributing to the high cost, as well as the acknowledged poor quality of DOD software include the proliferation of high order languages (HOL), inadequate software tools, and the widespread use of assembly languages in embedded computer systems. These factors have been recognized and policy initiatives taken in DOD Directives 5000.29 and 5000.31 to require the use of a higher order language (HOL) instead of assembly language in embedded computer systems and to achieve HOL commonality.

It was recognized that due to the nature of embedded computer systems that a usable common language may have specific requirements. To that end, a Tri-Service HOL Working Group, formed in January 1975, generated, via a continuous open forum, a requirements document that has now gone through five levels of refinement. The current version of the requirements document is known as "STEELMAN". Existing languages were evaluated against these requirements and it was determined that none were satisfactory for the long term. Further, it was concluded that a single language could suffice for all types of embedded computer systems. Existing baseline languages were selected, however, as potential points of departure (rather than starting "from scratch") following which four parallel competitive contracts were awarded for preliminary design of the common HOL (which is informally known as "DOD-1"). These designs were delivered on 15 February 1978 and are currently being evaluated.

Recently, three economic analyses were conducted to determine the potential payoff, DOD-wide, from the establishment of a common language. These analyses spanned the period from the early 1980's through 1999. The results vary as a function of the specific scenario selected which is based on a variety of factors such as time of introduction, rate of acceptance, R&D investment, etc. It was anticipated that benefits would flow from both commonality and new technical features. Commonality elements included training, support software tool focus, experience, communications, transportability, and increased hardware choice. Technology elements included impact on the design process, ability to handle non-standard I/O, real-time features, parallel processing, modularity, readability, maintainability, object-code efficiency, ease of programming, hardware independence, and error recovery. The first analysis was deliberately based on very conservative scenarios (e.g. no growth in software requirements). The cost benefits (over and above investment and in 1977 dollars) in this were 110 to 900 million dollars. The remaining two analyses were very close, with savings in the range of 12 to 24 billion dollars.

2. Motivation for this Paper

It is recognized that this language may have wide spread influence as it embraces participation from the three armed services, NATO and the European community. This recognition has led to leading international computer language designers contributing to the design effort. It is also recognized that various potential users of the language are concerned that the resulting language may not be applicable to their particular area and are further concerned that they may ultimately be forced via DOD policy to use the language where they feel it does not apply.

It is the objective of this paper to attempt to alleviate this concern within the telecommunications area by discussing the technical capabilities of the language relative to some of the major technical problems that arise in telecommunication system software designs. As an example, we take a 50 line message switch such that would be used in the worldwide defense communication system and design and code it in terms of the features that the common language will possess. In order to keep the discussion within a palatable scope, we will concentrate on a particular important issue and mention the others in passing. In that way we can treat the entire design process from a functional description right down to code in the language and include a discussion of the semantics of the resulting code.

3. The Functions of a Message Switch

It is our view that the language can have a significant impact on the design of a system; for that reason we do not attempt to duplicate any particular existing message switch design but choose rather to begin with a functional specification developed from the functions provided by existing designs. We develop this functional descriptive material beginning with a closed communications network of which several identical message switches are a part.

It is the primary purpose of the network to route messages from an originating subscriber to other subscribers in the network that are designated to receive the message by the originating subscriber. A message is considered to be a finite sequence of ASCII coded characters. The character is the atomic entity of the message (as opposed to the bit in other systems).

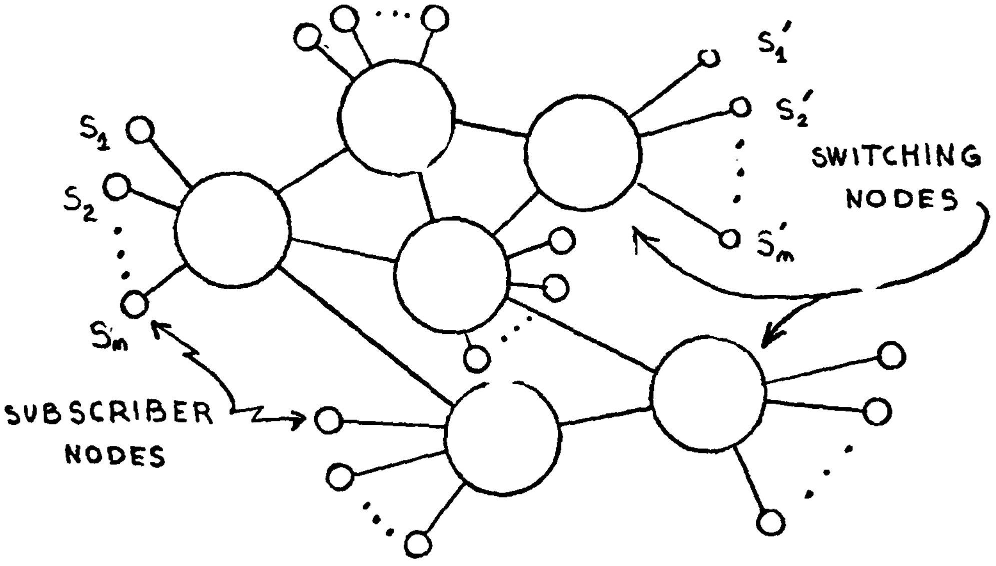


FIGURE 1

Consider the communications network of figure 1. It consists of two types of node interconnected by means of two types of lines. These entities of the network are described as follows:

a) Subscriber Nodes

A subscriber node consists of facilities for constructing and transmitting or receiving and interpreting a message. We have arbitrarily chosen subscriber facilities to be relatively primitive, like a common teletype which generates or receives a single character at a time.

b) Switching Nodes

A switching node consists of facilities to control the path any message takes through the network. It is the design of one of these nodes with which we are concerned. Each of the switching nodes can connect directly to up to fifty subscribers and up to three other switching nodes.

c) Subscriber Lines

Each subscriber line consists of a pair of transmitting and receiving wires to give full duplex operation. Transmission over these lines occurs a character at a time in an asynchronous manner. Acknowledgement to the sender is on a complete message basis. There are additional network control commands which we omitted for the sake of brevity and the fact that they do not raise interesting design issues.

d) Trunk Lines

Each trunk can be considered for logical purposes as bundles of subscriber lines. Messages are sent synchronously in 80 character blocks each of which is checked for parity and explicitly acknowledged or non-acknowledged by the receiving node. The trunk lines are used to connect a switching node to another switching node.

4. Message Flow in the Network

A message is originated by a subscriber who sends his message character-by-character to his local switching node. The local switching node absorbs these characters until it recognizes a valid message. If the message is syntactically correct, the switching node explicitly acknowledges this fact by sending an "ACK" message to the subscriber; if it is syntactically incorrect it explicitly non-acknowledges the message by sending a "NAK" message to the subscriber. At the point in time when the ACK is sent to the node transmitting the message, the receiving node accepts responsibility for the message. The fact that a node has responsibility means he in effect "owns" the message and must be able to reproduce it in its entirety. A node is not allowed to transmit a message until it has accepted responsibility for it. The switching node now selects, based on data local to itself, a trunk line on which to transmit the message (or a subscriber line if the destination is local to the switching node). In a similar manner, the "responsibility" is passed from node to node until the destined subscriber node accepts final responsibility for the message. Notice that an originator by receipt of an ACK only knows that the rest of the network absorbed his message; he does not know if the message ever reached its destination. It should also be noted that this concept of responsibility dictates that the store and forward concept of message handling will be used. We have chosen two types of I/O (subscriber and trunk) on purpose in order to raise the issues involved with low-level and non-standard I/O.

5. The Message Control

In order for a switching node to control the path any message takes through the system, the originating subscriber must append certain information to the message text. A typical message syntax is as follows where the ';' is used to separate syntactic entities of the message.

```
MESSAGE = SOM;PRECEDENCE;CLASS;TYPE;ROUTING INDICATORS:
          ORIGINATING STATION;MSG SERIAL NR;DATE TIME GROUP;
          SYSTEM ROUTING INDICATORS;TEXT;EOM
```

By convention we will refer to the syntactic entities preceding TEXT as "header data".

The switching node scans an incoming message which is defined as all characters between and including the SOM (Start of Message) and EOM (End of Message). Among other things (which we will ignore for the sake of discussion) the switching node scans the message for the routing indicators which are used to indicate where the message is to be sent. Contained in the files of each switching node is a directory as shown below. On recognition of a routing indicator, the node searches the directory for a matching routing indicator and if it is found, determines the line or trunk identification number listed for that routing indicator. This number indicates which I/O line is to be used to send the message. For example:

DIRECTORY

ROUTING IND	PRIMARY	SECONDARY
RUEDBIA	51	52
RUKLDAR	23	-
.	.	.
.	.	.
.	.	.

If an incoming message contained the routing indicator "RUEDBIA", the switching node would attempt to capture the use of line number 51; if it could not, it would attempt to capture line 52. The "secondary" entry is present for trunks where a destination can be reached by alternate paths. Thus "RUEDBIA", as indicated, could only be a trunk line. If the scanned routing indicator is not listed in the directory, the message is non-acknowledged and a control message is sent to the message originator.

6. Consequences of Local Control

A direct consequence of control local to a switching node is the fact that messages can be routed into orbit patterns within the network and never reach their intended destination. For that reason, we need a means of detecting orbiting messages and changing their routing. In the system under discussion we provide a human supervisor who has certain abilities to deal with this condition.

A second direct consequence of local control and the concept of message responsibility results if a switching node fails; there is no global view of the status of the system. For that reason, we record on magnetic tape, called an archive, each action that a switching node takes on a message. In addition, in order to meet the responsibility requirement each message is copied into the archieve. We then provide the human supervisor with the ability to retrieve messages and log items from the archive in order to establish a backup point and the ability to retrieve and reintroduce messages into this switching node.

7. Supervisor Functions

The human in the system must, therefore, be able to access the archive files to obtain log items and messages, to change the directory in order to re-route messages, and to act as a subscriber to his own switching node in order to send and receive messages from other supervisors and subscribers. By the latter capability he can develop a more global view of the network status.

8. Design Areas

In our discussion so far we have implicitly raised several areas of design, these are:

1. Memory Management - We show the requirement for magnetic tape and now suggest that due to the potential volume of traffic some means of secondary storage such as a disk is also necessary. If multi-media storage is required then so is a memory management system. An HOL applicable to this problem must, therefore, have the ability to express the code associated with memory management.

2. Input/Output - We have developed the need for both an asynchronous and (hence interrupt driven) character level input/output and a synchronous (hence timed) blocked (or card image) input/output. An HOL applicable to this problem must also possess the ability to express low level non-standard input/output.

3. Program Structure - We have not developed the notion of complexity of a switching node, but argue by example that the code for switches of this type is relatively large; a 50 line message switch currently operating in Europe much like the one described is expressed in 250,000 assembly level program statements. In order for anyone to be able to understand that system for change or upgrade purposes, some partitioning method is necessary. In fact, due to another aspect of the nature of message switch, the discussion of which follows, a clear partitioning or modularization is mandatory.

4. Parallel Processing - The inherent nature of message processing is such that many individual tasks are largely independent and the order in which the processing of these tasks occur is irrelevant. In fact, when arbitary, fixed order is imposed by the system, the results can be detrimental to the performance of the system. We need the ability to express the fact that such processes run in an order depending only on specified dynamic aspects of the system. We consider that such processes run in parallel or execute concurrently.

9. Applicability of Research & Development in Operating Systems

These four very important design areas of message switch design have been the subject of intensive research during the last 5 years under the computer science field of Operating Systems. An applicable piece of this work is the study of languages for operating system designs. The language that we will use is one specifically designed for the implementation of operating systems. Rather then discuss all the issues of system implementation languages, however, we choose to concentrate on the fourth item as it applies to the design of a switching node.

10. Parallelisms in a Message Switch

Consider the four subscriber nodes and single switching node shown in figure 2.

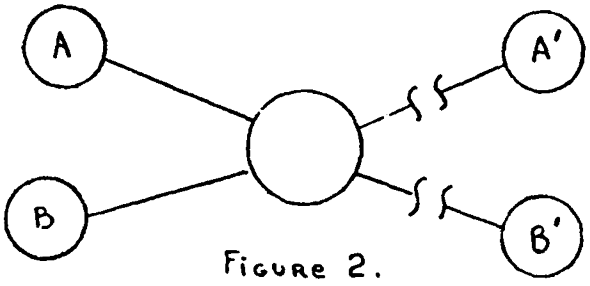


Figure 2.

The broken subscriber lines connecting the switching node to subscribers A' and B' mean that, relative to our discussion, an arbitrary arrangement of switching nodes could be inserted at these break points.

Suppose A wishes to send a message to A' and B wishes to send a message to B' at the same time and the two messages are independent. If these two messages were of the same precedence and classification then any order that the switching node chose to process them would be completely arbitrary. Logically, we would like the switching node to process these two messages in parallel or concurrently and not impose an explict order. If we had a switching node with many such messages arriving at the same time, we would like to consider that all messages are processed concurrently. There is inherent parallelism associated with message switching. Hence if one wants to capture the functions of a message switch in an HOL the language should be able to express the notion of parallel processing. In fact, within the processing required for each of these messages there are activities associated with each message that should also occur in parallel. For example, the scanning of the message characters for the routing indicators could be accomplished while the message is being stored in secondary storage.

11. If we had a system with infinite resources, independent parallel processes could in fact be allocated each system resource that they individually require, but such is not economic reality. Resources such as disks, memory and the CPU(s) itself must be shared among several parallel activities. For that reason, we need to express the concept of mutual exclusion. That is if the nature of a resource is that it can be used only by one process at a time, then once it is captured by a process we need a mechanism to exclude others from using it also. Implicit with the mutual exclusion concept is the necessity of scheduling which of the excluded processes will acquire the resource when it is eventually released.

A third design issue occurs for those parallel processes which are not entirely independent. Consider a process which needs the results of two independent parallel processes before it can continue. Some sort of synchronization is needed to signal when these results are available.

12. Language Features and Design Influence

We have found that the particular features for expressing parallel processes, mutual exclusion of shared resources, scheduling and synchronization have considerable influence on the system design. In fact the "wrong" features, or lack of appropriate features, in this area can actually hinder the design process. The language must provide the designer with the facility to express himself in a way that is natural to his application and should aid his design process, not hinder it.

For example, suppose one were designing a switching node such as the one just discussed and say that the system must run on a uniprocessor for which a multi-tasking operating system was not available. Suppose also that the available language did not express parallelism. Such is often the case for military systems. Then, because of the inherent parallel nature of the message switch, the designer would have come up with a design for sequentializing his tasks to run on the single CPU. This sequentialization method must be accomplished early in the design process, as it touches most aspects of the switch design. Projects designed in this way tend not to have clean modularization in the overall design. Consider for example, the addition of a new communications line with a different protocol. This addition may require significant changes in the definition and scheduling of parallel tasks; in fact the sequentialization method may not be generalized enough to be able to accomodate the additional tasks required. This can require a complete redesign of the sequentialization method and its communications with other parallel acting tasks. The effects then can affect many modules. Suppose, on the other hand, the language used in the design offered an appropriate facility for expressing parallism. Then these considerations are structured by the language and are more easily identified and isolated. If a language cannot express parallism, then the resulting designs are always contorted by the code used for mutual exclusion, scheduling and synchronization with the result that understandability of the program suffers with the consequence that management and maintenance problems inevitably result.

13. Designing for Parallelism

If one were to attempt to design a message switch, one usually starts with a given set of hardware which in turn dictates the operating system and languages that are available. We choose to call this collection of hardware, OS the language system. We will consider the following three types of language systems that the present day designer is confronted with relative to the system's ability to support switching system design.

1. There are no language features to express parallelism and there is no multi-tasking operating system. In this case, the designer must partition his problem into sequential routines and express them with an available sequential language (e.g. Fortran IV). He must then build the operating system functions for process control, mutual exclusion and interprocess synchronization. For efficiency reasons, and the fact that these functions are called often in the course of message switch operation (due to the highly parallel nature of the switch), he will most likely code these functions in assembler language. The resulting system program will look mostly like a higher-level language program with a few interdispersed assembler statements. The logical flow such an innocent-appearing program is not at all clear, however as the process control assembly statements are akin to the GOTO; where the program control is transferred at this point in the program is dependent entirely on dynamic conditions expressed in the assembly code. Under what conditions the program control returns is usually elusive.

2. There are no language features to express parallelism but there is a multi-tasking operating system. This case is identical to the former case except that the interdispersed assembler code is replaced by calls on the operating system. The development of a switching node using this system would be far easier than the first, however, the logical flow of the program again is unclear as the return from the calls on the operating system is dependent on the dynamic conditions of the system and the internal functions of the operating system. The designer now is at the mercy of the invisible internal design of the operating system. We have found that often these standard operating systems afforded generality at the price of efficiency. In other words, too many checks are performed at run time and too many registers are saved when a process context is switched in order to be safe. Driven by the need for efficiency, the designer is usually forced to modify the operating system which is usually written in assembler level code. He must then go through the process of understanding someone else's assembler level program, a most difficult task.

3. A language is available that has unstructured language features. This language is able to communicate with the operating system and configure tailored run-time support for the particular application program. There have been some experimental languages that have these abilities. They usually use low level primitives in the language such as the Dijkstra Semaphore for mutual exclusion and synchronization. This approach is better than the other two as it results in a tailored runtime support package for multi-tasking; however, the semaphore allows few compile time checks when used for both mutual exclusion and synchronization; it can only be checked at compile time to see if it is a type semaphore. More sophisticated compile-time checking is necessary in order for a higher order language to be effective for the following reason. Those errors not detected at compile time result in run-time errors. When a sequential program is checked for run-time errors, the designer can repeatedly put in a set of fixed inputs and observe the state of the program variables as it executes. For a set of fixed observation points during the course of this running, the state of these variables will be the same each time the program is run. He can hypothesize what these states should be, make a run-time observation, resolve the difference between his hypothesize and the observed results, make a change and repeat the process. This debugging procedure CANNOT be followed for an interactive parallel program as the behavior of the program variables is dependent on the dynamics of the system as a whole. If such a program is repeatedly run with a fixed input, the state of its variables will not always be the same at the fixed observation points; the variable states are non-deterministic in general. For this reason, we are motivated to consider more structured language features so that more meaningful compile time checks can be made.

14. Structured Parallel Processing - DOD-1

The IRONMAN requirements specify the need for structured parallel processing features in the language. At this time the actual features of DOD-1 are unspecified. For that reason, we choose to use the features of an available language called MODULA 5,6,7, a parallel processing language based on PASCAL [4],knowing that the features that it offers cannot escape the attention of the DOD-1 language designs. The following features of MODULA are particularly interesting with regard to the design of our switch.

1. The MODULE - This is an entity in the language used to encapulate programs and data and restrict their use by other programs. A module is a collection of constant, type, variables, and procedure declarations, called objects. None of the objects declared within a module are accessible from outside the module unless the particular object is explicitly exported by a statement within the module.

2. The PROCESS - The process is like an ALGOL procedure except that when called it executes in parallel with the calling program. The process is the means for expressing concurrent activity.

3. The INTERFACE MODULE - The interface module is a special case of a MODULE. In addition to the features of a module, the interface module is based on the idea of the Hoare Monitor[3] and allows only a single program to use its exported objects at a time. This is a structured mechanism for mutual exclusion. Programs that attempt to use an exported object that has been acquired by a previous user are automatically queued up in a prioritized FIFO queue.

4. The DEVICE MODULE - The device module is a special case of an INTERFACE MODULE. It differs only in that it may contain a device as one of its objects. Wirth in the design of MODULA recognized that devices such as line printers can logically be considered as processes running in parallel with their activating program. The reason for a distinquished type of module to encapsulate a device process is that communication to these devices are usually by command codes, bit vectors and status registers all at machine code level. In order to isolate the areas that machine code could be used in MODULA, machine level coding is only allowed within a DEVICE MODULE. Within this module we can identify particular machine registers by name and manipulate their individual bits.

5. SIGNALS - A signal is a type in MODULA that is used for process synchronization. A process can send a signal, wait on a signal or check if any other process is waiting on a signal. Unlike a semaphore, a signal has no memory and therefore is simpler to implement. If a send operation is performed on a signal and no process is waiting on that signal, no action is performed. We now demonstrate the facility of these concepts in a design of a message switch.

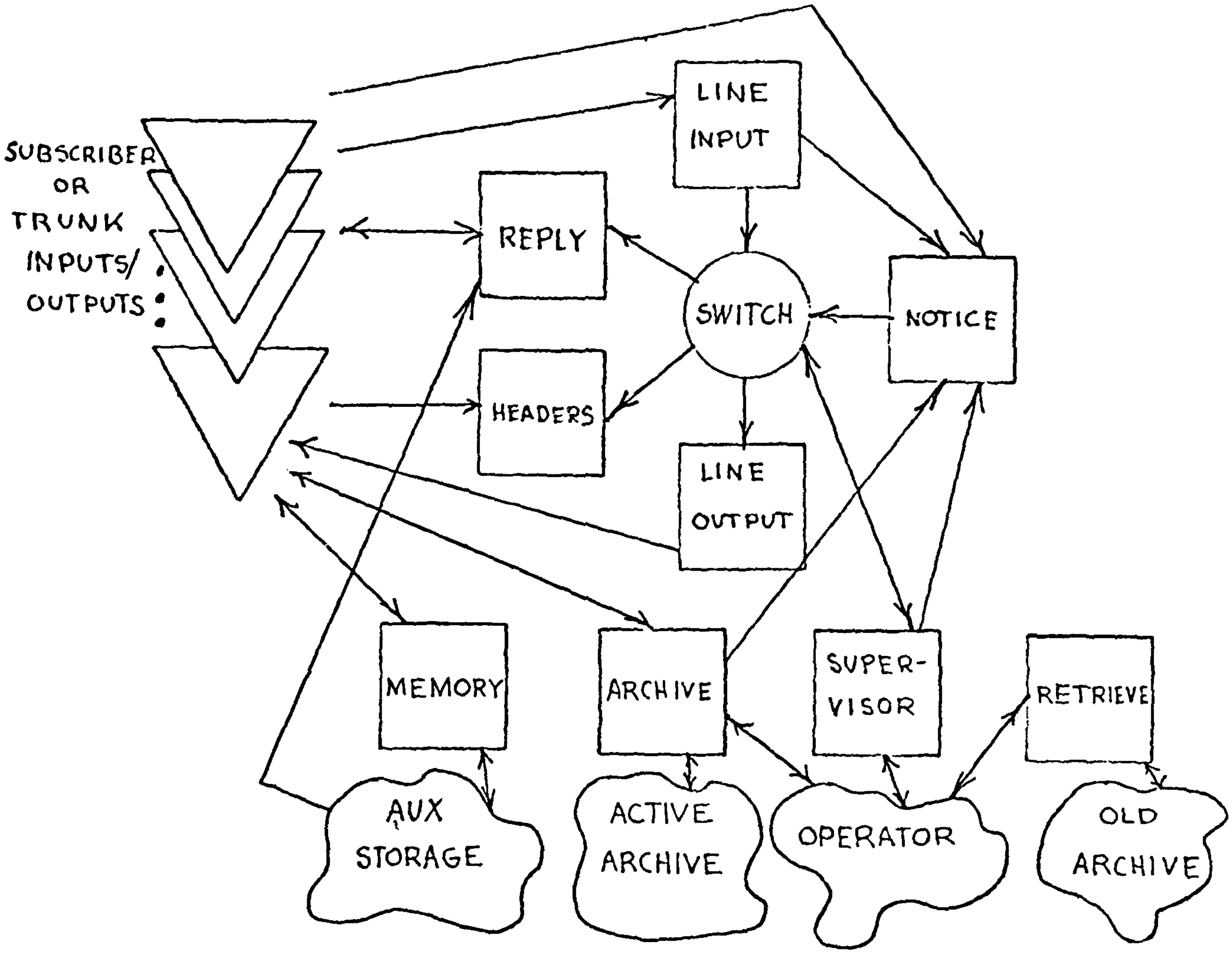


FIGURE 3

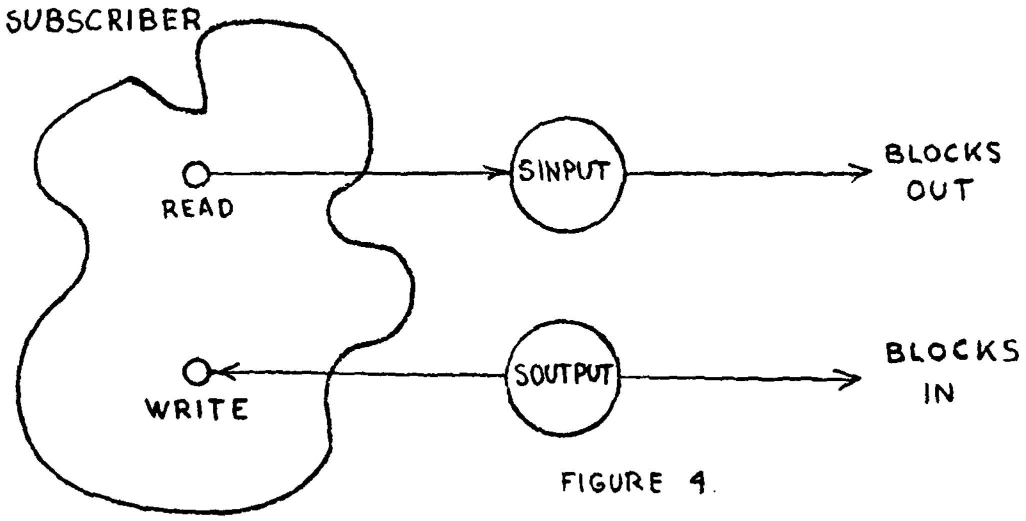


FIGURE 4.

15. The Switching Node Components

The switching node components as expressed in MODULA entities are shown in figure 3. In that figure, the circle labeled"SWITCH" is a process called by the main program that attempts to execute all the time. The squares labeled "NOTICE", "ARCHIVE", etc are all interface modules resolving contention for their objects by the numerous callers that are indicated by the directed lines. The ink blot shape entities are device modules: e.g. "AUX STORAGE" contains a disk, "OPERATOR" a teletype, etc. The triangles are not of themselves entities of the language, but each contains the language entities shown in figure 4. There is a separate triangle for each subscriber in the system (we have 50). They are identical except for their names. There is also a separate triangle (that differs from the subscriber triangle) for each trunk line connected to the switching node (we have 3). We point out that figure 3 is a complete description of a design for the switching functions that we have previously discussed; it is a complete description of a whole switch. To convince the reader, we will now show how these entities cooperate to input and output messages. To show the power of this approach we will then focus on the operations contained within the triangle and finally within a device module.

16. Inside a Subscriber Triangle

The entities of a subscriber triangle are shown in figure 4. A subscriber triangle consists of a device module called SUBSCRIBER which has exported the ability for an outside program to read a single character or to write a single character. In addition, it has two processes, SINPUT and SOUTPUT. SINPUT continually attempts to read a character, perform some lexical processing, and buffer the characters into 80 character blocks. SOUTPUT continually attempts to input 80 character blocks and write a character at a time into SUBSCRIBER. We will discuss what it means to read and write a character into "SUBSCRIBER" shortly, but first let us consider message processing by the switch as a whole.

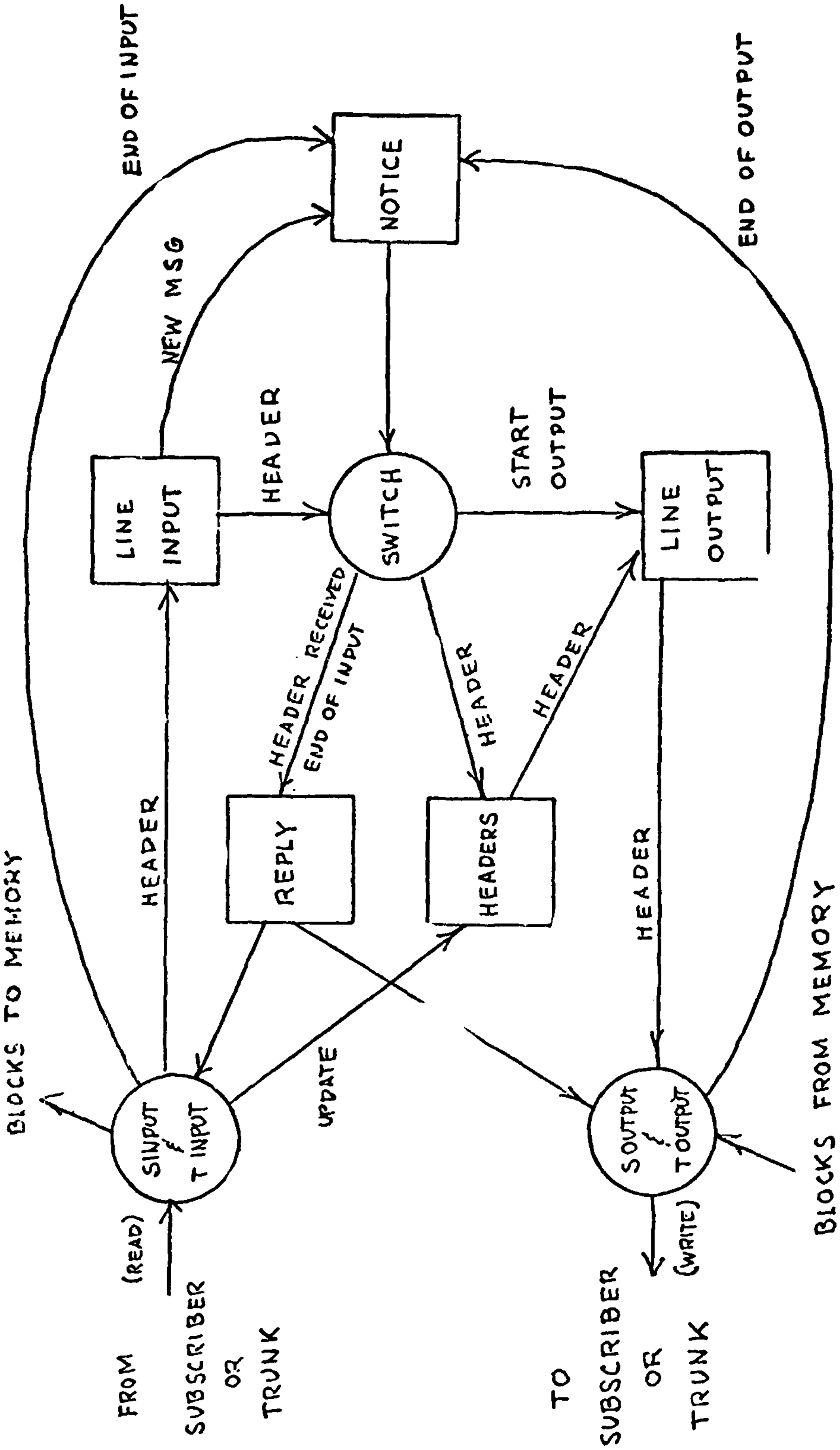

NOTICE
END OF INPUT
NEW MSG
END OF OUTPUT
LINE INPUT
HEADER
SWITCH
START OUTPUT
LINE OUTPUT
HEADER RECEIVED
END OF INPUT
HEADER
HEADER
HEADER
REPLY
HEADERS
HEADER
HEADER
BLOCKS TO MEMORY
BLOCKS FROM MEMORY
UPDATE
SINPUT & T INPUT
SOUTPUT & TOUTPUT
(READ)
(WRITE)
FROM SUBSCRIBER OR TRUNK
TO SUBSCRIBER OR TRUNK

FIGURE 5

17. Control of the Reception and Transmission of Messages

Consider the switch components shown in figure 5. This is a subset of those shown in figure 3 in order to focus our attention on message control. Let us trace the flow of a message being input from a subscriber and output to another subscriber. SINPUT of figure 5 is continually attempting to extract characters from a particular subscriber device module. SINPUT is the same SINPUT shown in figure 4; there is one for each subscriber line. SINPUT makes a call on SUBSCRIBER to read a character. We shall shortly see that if there is none, SINPUT will be put in a wait state by relinquishing control of the CPU. SINPUT will be subsequently given control of the CPU if SUBSCRIBER can produce a character. If SUBSCRIBER can produce characters, SINPUT attempts to form the first 80 character block and, at the same time, extracts header data from the message. SUBSCRIBER generates an internal representation of this header data and places it in a buffer within the interface module LINE INPUT. LINE INPUT now knows that SUBSCRIBER is receiving a new message and generates an internal code "NEW MSG" which he places in the interface module NOTICE. NOTICE serves as a mailbox for the process SWITCH who is, in reality, the central manager of this switching node. SWITCH is continually looking into NOTICE for directives. He finds "NEW MSG" which causes him to extract the header information from LINE INPUT. SWITCH as system manager keeps the log of all actions the switching node takes on each message. He enters the time of receipt, header data, and the fact that complete message receipt is pending in his log. He then checks the routing indicator with his local directory. If the indicator is valid he sends an internal message to the interface module REPLY, indicating that the header data was good. He then stores the header information in the interface module HEADERS. In parallel with all this activity, SINPUT is sending 80 character blocks of data to the MEMORY interface module where it will subsequently be stored on AUX STORAGE. A recognition of "EOM" by SINPUT triggers a message to NOTICE that the end of this message has arrived. SWITCH notes this fact in the log, accepts responsibility for the message, posts an "ACK" message to the interface module REPLY who sends "ACK" to SOUTPUT who transmits it via SUBSCRIBER to the message originator.

As soon as SWITCH accepts responsibility for the message he looks up in his log where the message is to be sent (there may be more than one addressee) and notifies the interface module LINE OUTPUT to insert this message in the appropriate line queue (located within LINE OUTPUT). SWITCH instructs HEADERS to send the header information for the message to LINE OUTPUT. When there are no higher priority messages to be sent to the designated line, LINE OUTPUT sends the header information to the designated SOUTPUT who forms syntactically correct header data and attempts to transmit this data character-by-character to the appropriate subscriber. After the header data is sent, SOUTPUT retrieves the subsequent message blocks from MEMORY. When the last block has been transmitted, SOUTPUT posts a message in NOTICE that the "end of output" has been reached. SWITCH enters a log item to that effect.

Thus a description of switch functions can be held in terms of the language entities. For completeness we will now take a closer look at what is typically inside each interface module by looking at the code for SUBSCRIBER.

18. The Subscriber Device Module

The code that enables the device module to input a character and send it to the SINPUT process is shown below. We have omitted the code that enables this module to accept a character from SOUTPUT and transmit it to the subscriber device for brevity. The line numbers are not part of MODULA but are for our reference purposes only.

```
  DEVICE MODULE SUBSCRIBER;

10    DEFINE READ, WRITE;

20    VAR INR, OUTR, NRF : INTEGER;

30        NONFULL, NONEMPTY : SIGNAL;

40        IRBUF : ARRAY 1 : N OF CHAR;

50        similar output buffer declarations

60    PROCEDURE READ (VAR CH : CHAR);

70      BEGIN

80         IF NRF = Ø THEN WAIT (NONEMPTY) END;

90         CH := RBUF  OUTR :

100        OUTR := (OUTR MOD N) + 1;

110        DEC (NRF);

120        SEND (NON FULL);

130     END READ;
```

```
PROCEDURE WRITE (VAR CH : CHAR);
                 .
                 .
                 .
END WRITE;
PROCESS INPUT;
    VAR CH : CHAR;
    BEGIN
      LOOP
        IF NRF = N THEN WAIT (NONFULL) END;
        start read into BUF  INR ;
        DOIO;
        INR := (INR MOD N) + 1;
        INC (NRF):
        SEND (NONEMPTY);
    END
END INPUT;
PROCESS OUTPUT;
      .
      .
      .
END OUTPUT;
BEGIN
    INR := 1;
    OUTR := 1;
    NRF := Ø;
    INPUT;
    similar output initializations
END SUBSCRIBER;
```

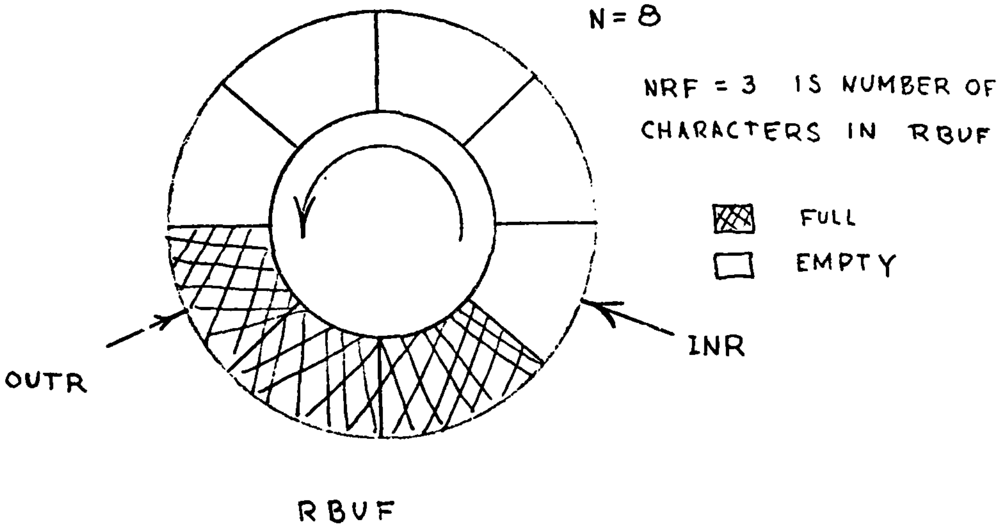

N = 8
NRF = 3 IS NUMBER OF CHARACTERS IN RBUF
FULL
EMPTY
OUTR
INR
RBUF

FIGURE 6.

This is a MODULE so that the internal operations are hidden from the other using programs and the ideas of "read a character" or "write a character" can be abstracted from internal operations and exported to the using program. It is an INTERFACE MODULE so that only one outside process can obtain the ability to read or write at a time. It is a device module because it contains a teletype (connected by a long subscriber line) that can be considered an independent process. In line 10 we explicitly export the ability to read and write with a DEFINE statement. Line 20 declares and lines 310 through 330 initialize the circular buffer as pictured in figure 6. READ is declared as an exported procedure (which we know from previous discussion is called by SINPUT) which passes by reference a variable named CH of type character. Lines 160 - 270 declare a process INPUT which is called once in line 340 and runs forever. If we start the system from a "cold start", SUBSCRIBER will initialize the circular buffer and start the process INPUT. The LOOP statement of line 190 states that lines 200 - 250 repeat forever. Within the loop, INPUT looks to see if the buffer is full (line 200) if it is, the process waits on a signal called NON-FULL. Execution of this wait means, INPUT relinquishes control of the CPU and gets inserted in a queue associated with the signal NONFULL. In other words, the buffer is full so INPUT cannot insert anything in it. Suppose the buffer was not full, then INPUT would "start a read" into the circular buffer. The english was substituted for an actual device code which would be placed in a device command register at this point in order to avoid discussion of a particualr machine's I/O. Also at this point we would assign a register to the interupt vector if I/O is controlled that way. Line 220, DOIO, means wait for the device to interrupt. That is, relinquish control of the CPU and wait for an interrupt from this device. Note that all subscriber modules spend most of their time at this point. If the subscriber presses a teletype key, the awaited interrupt will appear. Lines 230 - 240 insert the incoming character into the buffer; line 250 announces to any waiting processes that the buffer is now not empty.

The Procedure READ (line 60 - 130) is called by the external process SINPUT. Line 80 of this procedure causes READ to be suspended on NONEMPTY of the buffer is empty. Suppose we have just put a character in the buffer via INPUT. Then READ would remove it (lines 90 - 110) and send a NON-FULL signal to indicate that at least another character can be put in the buffer. (INPUT may be waiting for this signal on line 200.)

This program illustrates the fact that we have been able to synchronize the two independent processes by means of signals and a buffer, and to isolate that mechanism from the rest of the system by use of the module and its explicit export mechanism.

19. Conclusions

Referring to figure 3, we have been able to express the structure of a switch at a fairly high level using entities of an appropriate language. Note that the addition of a subscriber or trunk line to this system is trivial as is the implementation of a new protocol. It merely means the addition of another triangle or the design of a new triangle respectively.

We have stressed that language expressiveness in the area of parallel processing is important to the design of a message switch and claim that this can be generalized to include a large percentage of the telecommunication processing at large. We must point out however, that other language areas that were totally absent from our discussion are also very important and can have a significant effect on telecommunication software design. For example, IRONMAN requires EXCEPTION HANDLING (what does a system do to recover when something goes wrong at run time) REAL TIME PROCESSING (access to system clocks), SEPARATE COMPILATION (for building large systems).

By using the high level constructs of MODULA we coded all the entities for a 50 line message switch in a total of some 1500 lines of HOL code. The reason for this compact expression is the fact that message switching abounds with parallel activity and that the language features were well matched to this activity. We suggest that this fact alone ought to generate interest in the applicability of DoD-1 to telecommunications areas. We, therefore feel that the technical features found in the new Department of Defense Common Language will be an asset to the design of telecommunications software.

20. Acknowledgements

This work was performed at the Center for Tactical Computer Sciences (CENTACS) at Ft. Monmouth, NJ; the need for such a project was suggested by the Army Communications Command at Ft. Huachuca, AZ. The project was conducted by author, Dr. Serafino Amoroso of CENTACS and Prof. Gregory Andrews of Cornell University [1,2].

21. References

1) Andrews, G.R. "Modula and the design of a message switching communication system". Technical Report 78-329, Department of Computer Science, Cornell University, January 1978.

2) Gregory R. Andrews, "The Design of Parallel Systems: An Application AND Evaluation of Modula", Cornell University Tech Report TR 78-330, January 1978.

3) Hoare, C.A.R. "Monitors: an operating system structuring concept". Comm. ACM 17, 10 (October 1974), 549-557.

4) Wirth, N. "The programming language Pascal". Acta Informatica I (1971), 35-63.

5) Wirth, N. "Modula: a language for modular multiprogramming". Software - Practice and Experience 7 (1977), 3-35.

6) Wirth, N. "The use of Modula". Software - Practice and Experience 7 (1977), 37-65.

7) Wirth, N. "Design and implementation of Modula". Software - Practice and Experience 7 (1977(, 67-84.

An Integrated System of Tools to Support The DOD Common Language

Dennis J. Turner

CENTACS

Support software tools are typically a collection of independently designed programs which support no specific higher order language and which provide a non-uniform and often unfriendly user interface.

This paper described an integrated system of cooperating tools which supports DOD common language program development in a friendly and powerful environment.

AN INTEGRATED SYSTEM OF TOOLS TO SUPPORT THE DOD COMMON LANGUAGE

Dennis J. Turner

US Army Communications Research
and Development Command
Center for Tactical Computer Systems
Software Engineering Division
Fort Monmouth, New Jersey

Abstract

Support software tools are typically a collection of independently designed programs which support no specific higher order language and which provide a non-uniform and often unfriendly user interface.

This paper describes an integrated system of cooperating tools which supports DoD common language program development in a friendly and powerful environment.

Motivation

Software problems associated with military embedded computer systems are well known. "Excessively expensive, untimely, difficult to maintain, non-responsible to user requirements, inflexible to change, unreliable..." are all common criticisms of most systems developed to date. The inadequateness and lack of standardization of the Program Development and Maintenance Environment (PDME) has contributed significantly to the perpetuation of many of these problems.

One aspect of the PDME involves the software tools which can provide support in virtually all aspects of the programming process. However, support tools have fallen far short of their potential value. It is worthwhile to consider some of the problems which have impeded the progress of support tools to date.

1. Non-Reusability

The proliferation of programming languages and computers has given rise to a situation where most developers find themselves with a unique combination of language and host/target processors. Consequently, existing tools cannot be used and they must develop their own.

2. Minimal Tools

Because tools have been developed by the project for the project, resource constraints (manpower, funding, time) have usually compromised tool development in favor of the embedded application. The result is that only the minimum number of essential tools are developed. There have been cases where a language processor (compiler/assembler) was the only tool available.

3. Lower Standards

Because the tools are not the focus of a development (the embedded system is) and are often not even deliverables under a contract, a significantly lower level of quality results. Tools designs and implementations are typically ad hoc and poorly documented.

4. "Bag of Tools" Approach

Tools have usually been viewed as semi-independent functions. Their implementations reflect that view and the result is a "non-uniform" often "unfriendly" user interface. A command to one tool is often very different than the command to another tool, even when the objective is identical. For example, END, STOP, EXIT, HALT... might appear across several tools as the command for terminating the current activity. A "uniform" interface would allow the use of a single command, independently of the tool being accessed.

Some tools are designed in a way that makes their use extremely awkward. Command strings like **/% ... and the absence of feedback prompts to the user, make a tool very clumsy to use and present a less than "friendly" interface.

The "bag of tools"approach also causes common sub-functions across tools to be unrecognized and the result is a considerable amount of unnecessary duplication.

5. Language Independence

Well intentioned tool designers often seek to maximize the utility and flexibility of a particular tool and strive for language independence rather than support for a specific language. Their objectives have been narrowly focused on the tool rather than the larger environment of which that tool is a part.

Each of these problems has contributed significantly to the present day software dilemma of embedded computer systems. If they are to be eliminated in the future, a new view of the entire PDME must be taken.

A System View

It seems clear that if a dramatic impact on the software problems is to be achieved, the PDME must be viewed as a system and standardized to the maximum extent possible. One unifying aspect of the environment is the language itself. The adoption of a standard DoD common language (currently underway) is a major step in the right direction. But, it should be recognized that the language is only one feature of the PDME and we must look beyond the language to the entire environment.

Another aspect of the PDME involves the support software tools and it is this area which will be explored in the discussion which follows.

Tool Objectives for a DoD Common Language Environment

In order to develop a unified concept for support software tools, consideration must first be given to the following objectives.

1. Support of the DoD Common Language

Because the DoD Common Language will be defined and adopted in the near future, we would like to consider the use of support tools which complement and strengthen the use of that language and which enrich the entire PDME surrounding it. Whenever the dilemma of generality versus stronger support for the language arises, the later should be chosen.

2. Integrated System

A collection of tools ought not be viewed as independent entities ("bag of tools") but, rather as an integrated system of cooperating functions which are designed to provide a uniform and friendly user interface. The tool system should encourage and, to the maximum extent possible, enforce "good" programming practices.

3. Reusable

The tools should be written in the DoD Common Language in order to maximize their potential for transportability. The DoD Common Language Compiler should be structured and include a well-defined code generation module. The tools can then be moved to a new host processor by designing an appropriate code generator and by performing a self-compilation bootstrap procedure. Tools written in the DoD Common Language will also be more easily maintained and controlled.

4. High Quality Design and Implementation Standards

Unlike efforts of the past, the tools should be designed and implemented in accordance with state-of-the-art software engineering standards. They should be viewed as having a potentially long life cycle and should be developed with a high level of quality.

5. An Evolving System

Instead of attempting a quantum leap, the initial tool system should include only a modest set of known powerful tools which encourage convenient use. As tool requirements become more clear through use, and as the state-of-the-art advances, new tool functions can be incorporated into the system.

6. No Duplication

The tool system should be modular and structured in a way that encourages shared access to common subfunctions. Duplication of code across functions should be minimized.

7. Configuration Management

The tools which are used to produce an embedded system must be available throughout the entire life cycle of that system. Archival tool storage and upwards compatibility of tools must be carefully controlled in order to satisfy this requirement. To the maximum extent possible, this process should be automated by the tools themselves.

An Integrated System

We will now proceed to characterize a tool system that satisfies the objectives stated above.

The objective of integrating a collection of cooperating tools suggests a structure outlined graphically in figure 1. Here a central manager controls and monitors tool access. The major tool functions call upon various primitive functions for assistance and activities occur based upon information which is stored/retrieved via a common data base. This structure lends itself to modularity, cooperation and integration. It also provides the mechanism for elimination of duplication across major tool functions.

As an example, consider a compiler for the DoD Common Language as F_1 in figure 1. It should be structured in a way that not only allows information about a particular compilation to be available to other tools, but, also allows primitive tools used in the compilation process to be used by other major tool functions. Figure 2 illustrates such a structure. Control resides at the major function level and compilation proceeds as a series of accesses to the primitive functions. The lexical analyzer and the parser, comprising the compiler's "front end", could also be used by other major tools such as a language dependent editor. Information stored in the data base is also available for reuse. Symbol table and optimization data could be accessed by a symbolic debugger. Knowledge of the intermediate language would be of benefit to a symbolic program executor. The set of primitive functions, and the data base information shown is not intended to be exhaustive but only to illustrate the compiler concept being suggested.

It is not difficult to envision how this same structure could be applied to each of the major functions of a tool system. The key concept is the mechanism for sharing programs and data.

The initial tool system might consist of the following major functions:

A text editor strongly focused on the DoD Common Language for entering and modifying source programs. The editor would be language structure oriented (rather than line oriented) and would perform elementary lexical and syntactical checking.

A DoD Common Language Compiler which accomodates the language requirements and whose design is modular and compatible with the structure illustrated in figure 2.

A configuration/module manager which oversees the activities of module generation, interconnection and maintenance in an environment strongly oriented toward the language.

A linkage editor which integrates separately compiled modules into larger units.

A symbolic program executor (static debugger) which provides statistical information based upon the symbolic execution of a source program.

A collection of Documentation Aids which assist in the generation and modification of software documentation and which supports the user in selectively reading it.

A dynamic symbolic debugger which supports program debugging in the target environment at the DoD Common Language level.

Concluding Remarks

For many systems initial program development represents only a small fraction of overall life cycle costs, the major cost being that of maintenance. The availability of a PDME such as that described above would enable maintenance to occur in the same environment as that of development. The complete history of a program would be available to implement and validate changes. The view of maintenance as an extension of development (and not as a detached exercise in field "patching") should be a goal of the future.

It should be recognized that the main thrust of this paper has been to call for the application of the same state-of-the-art principles to tools as are applied to other software. A "system view" of tools is not a unique notion, only a late one.

Summary

A standardized system view of the PDME must be taken if the problems associated with embedded computer systems are to be successfully solved.

A standard DoD Common Language in combination with an integrated system of language dependent tools will have a dramatic impact on the present day software problems.

Acknowledgement

The author wishes to acknowledge the contributions of Dr. A. Nico Habermann (Carnegie-Mellon University) and Dr. Dewayne Perry (Pegasus Systems).

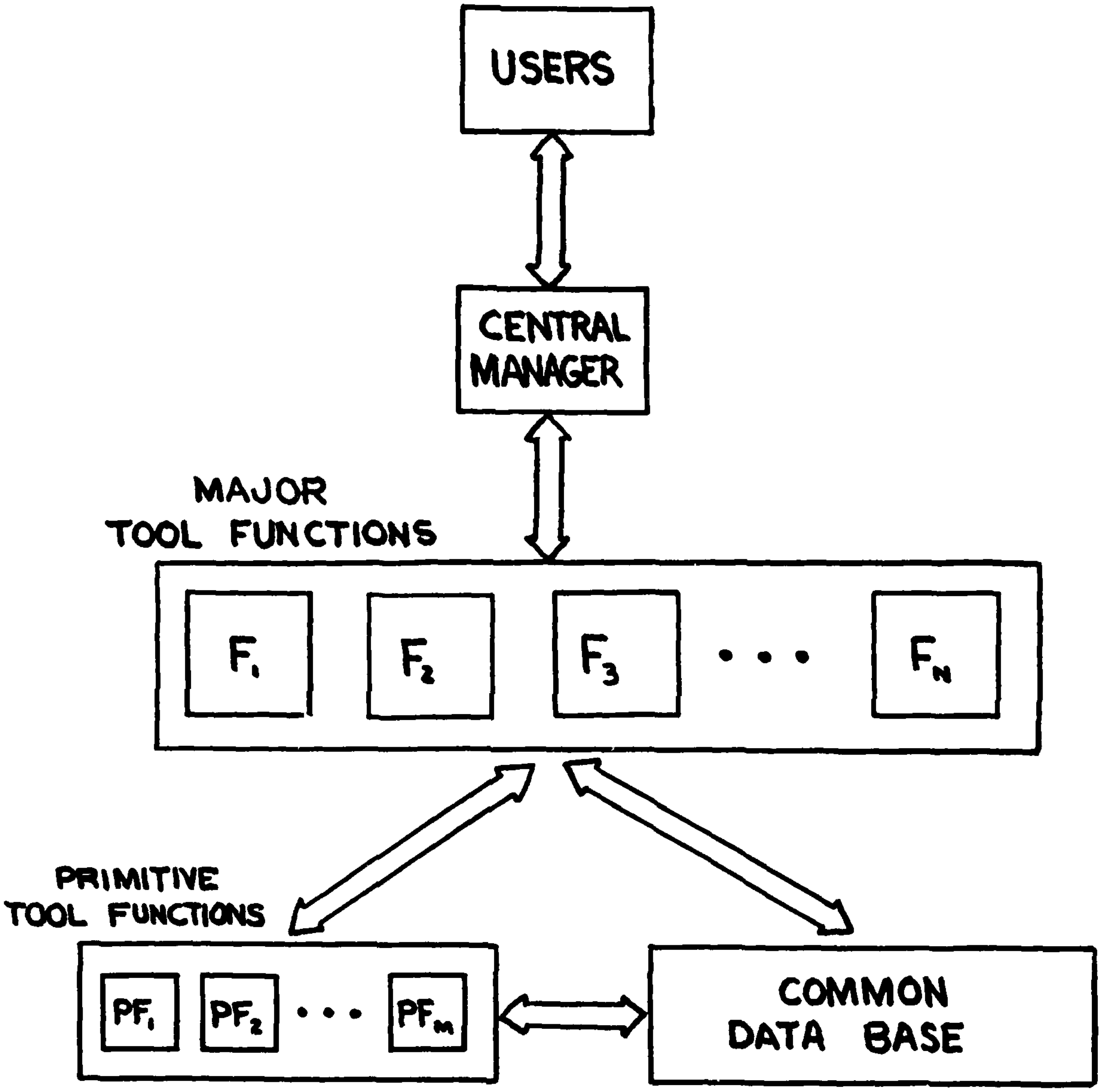


FIGURE 1
INTEGRATED SYSTEM OF TOOLS

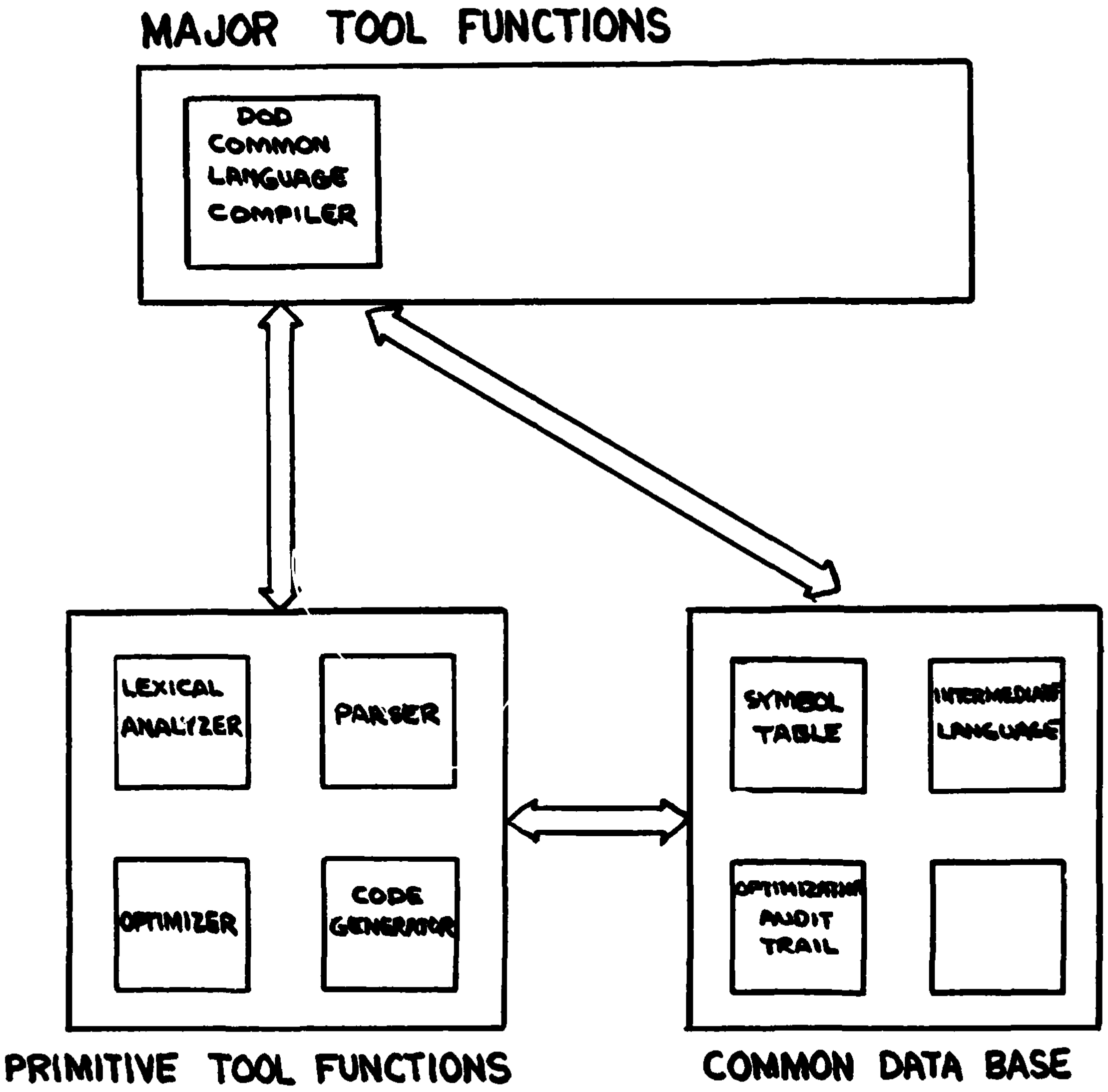


FIGURE 2
COMPILATION IN AN INTEGRATED SYSTEM

REQUIREMENTS II

John Mitchell
AIRMICS

REQUIREMENTS II

SESSION CHAIRPERSON: John Mitchell

AIRMICS

SESSION SUMMARY

The objective of this session was to present user experience with three software requirements systems which have been designed to meet the objectives of ensuring that requirements are complete, consistant, and unambiguous.

Much attention has been given to tools that are used to enhance our ability to program a solution to a problem. New and better programming languages are designed, structured programming concepts are implemented, etc. However, these techniques are of little value if the wrong problem is being solved.

Despite the existence of tools for aiding requirements analysis, their use is not yet widespread. This session examined three such tools and presented a summary of their use in various types of software development. We showed that the existing tools can be used, are being used, and that the use of any of these tools is far superior to the manual methods of developing, analyzing, and documenting system requirements.

Dr. Pei Hsia, University of Alabama, Huntsville, presented an overview of existing software requirements systems, examine different approaches to developing requirements, and present future research directions.

Mr. Charles Everhart, Teledyne Brown Engineering, presented the Input-Output Requirements Language (IORL).

Mr. Mac Alford, TRW, presented the System Requirements Engineering Methodology (SREM).

Dr. Paul Merrithew, LOGICON, presented the Computer Aided Design and System Analysis Technique (CADSAT) which is based on the Problem Statement Language and Problem Statement Analyzer (PSL/PSA).

DEVELOPMENTS AND PROBLEMS IN SOFTWARE REQUIREMENTS

Pei Hsia and Bill Buckles

SECTION I. INTRODUCTION

The "software crisis" is a term expressing the conclusion drawn form culminated experiences and reports on spiraling software costs, uncertain software quality, and frequent schedule and cost overruns of software projects. Many researchers are engaged in finding the cause and cure of this crisis. So far they have revealed many different causes but no cure. During the present soul-searching period, they have come to realize and accept three important facts: (1) there is lack of an understanding of software; (2) that "software is soft" is a misconception; and (3) there is an inherent technology push in software.

The lack of understanding about software is very well demonstrated by an inability to define and measure such software terms as reliability, complexity, and quality. Many such software terms are directly adopted from the hardware field. Although professionals in the field have similar conceptions, they are unable to reach a commonly accepted definition for them. This confusion is reminiscent of the early histories of many fields. The medieval cathedral builders, for example, labored without an understanding of material stress analysis. Though many of their structures collapsed, enough survived to demonstrate the viability of large architectural endeavors. The same way can be said of the early software projects. Only through diligent research and maturity will the true identities of the relevant software properties be unveiled and ways devised to measure them.

One of the most important lessons learned is that "software is not soft." Given a fixed hardware configuration, one cannot always assemble a set of functions to be accomplished by software that meet desired performance standards inherent in the environment. In other words, the theoretical concept that "software can do anything"* is not at all true if limited resources and expected performance are set in advance. Although the original purpose of the term software was only to distinguish it from hardware, the "ware" has been associated with "soft," "pliable" functions. This has caused many problems. For example, the practice has been to "shoe-horn" the remaining functions not easily incorporated by hardware into software, because software is "soft" and can accommodate. Many projects using this shoe-horning software practice had to be either abandoned or drastically revised. For the projects that were successful, one can use Figure 1 [1]

*This is a derived concept from the generally accepted hypothesis that any procedure can be realized by a Turing machine. Therefore, "anything" used in the sentence should be qualified by "that is doable through a procedure." But in a Turing machine, unlimited resource (tape) is used and timing is not a consideration.

to explain the cost of software resulting from the squeezing effect. This figure indicates how costly it is to squeeze too many functions into software, if it can be done at all.

The inherent technology push in the software industry, the trend toward larger and less numeric based projects, is evidenced by the ever increasing complexity of software. Although a formal quantitative definition of software complexity does not exist, there is a qualitative notion of the term. (Software complexity is different from computational complexity, which is a well defined term [2] .) An analogy can be drawn between housing construction and software development. The professional builders constantly build houses to the same blueprints. The only difference between houses is their location. The professionals in the software field seldom have the opportunity to develop the same product more than once. This is because a software product can be copied; once a FORTRAN compiler is developed, an exact copy of it can be used on all the computers with the same model number. Thus, in the software library, all development projects are untried projects. This phenomenon makes software a one-design, one-development industry, whereas housing construction (and many other industries) is a one-design, multiple-development industry. All one-design, multiple-development industries have a naturally built-in feedback system to check design and development methodologies. The one-design, one-development industry has a much broader domain to search for good design and development methodologies and much uncertainty to combat when achievement of a viable approach is claimed.

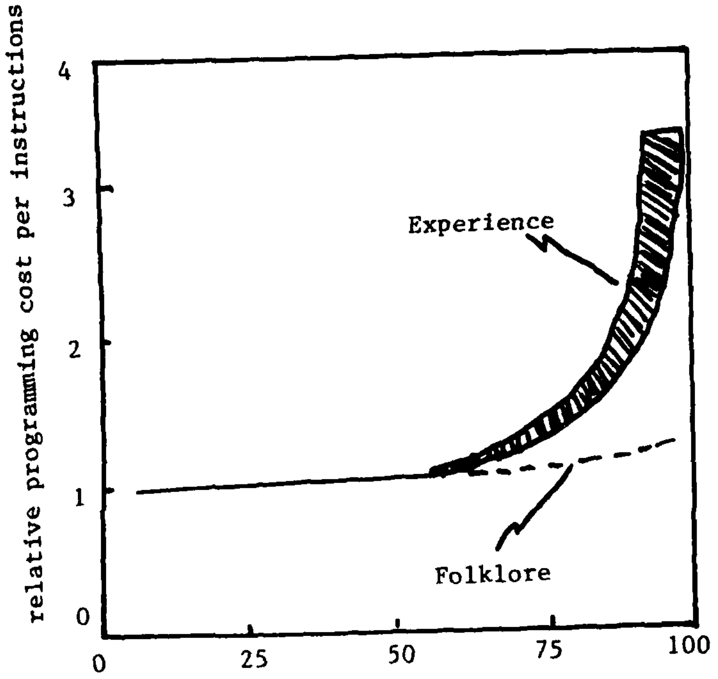


Figure 1. Software Shoe-Horning Aftereffect

SECTION II. THE ROLE OF REQUIREMENTS

It is obvious that improvements in any one of these three problem areas cannot fully be attained through more productive implementation methods. The most important problems lie in the area of software requirements [3] . What are requirements? Simply stated, one can say that software requirements are a statement of need -- a set of statements that describe the needs.

One more substantive definition of requirements states that requirements consist of any information generated prior to and in support of implementation [10] . That is, any document, either formal or informal preceding implementation but to be used in the subsequent design or implementation step is part of the body of requirements. Although this definition has some merit, we believe the following definition will be more useful in focusing attention on the problem areas in software.

> Software requirements consist of all knowledge concerning a problem at a given stage of development that has not yet been incorporated into an algorithm.

This definition enforces a "conservation of information" as indicated by Figure 2. As development progresses, the information simply changes form. The advantage of this definition is that it tends to direct attention to the area most in need of new methods and is less arbitrary than the former one. A computer code is, after all, an algorithm and differs from previous statements of the algorithm only in form, not substance. Thus, any statement of the algorithm is (or should be) equivalent to the coding.

But again the question: Within this framework, what are requirements? To be specific, software requirements can be classified into three categories: functional, performance, and resource. To obtain software requirements, one has to undertake a translation process that takes operational (or applicational) requirements as inputs and gives a statement of the functions to be performed and how well they should be performed as the output [4] . The functions are then allocated to different parts of a system within the resource limit to achieve the desired performance. This allocation process involves a considerable design tradeoff and feasibility analysis. The functions allocated to software and their corresponding performance and/or resource characteristics become the set of software requirements. Functional requirements dictate the set of functions that the software is to achieve. Performance requirements set a minimum standard on how well (accuracy, timing, etc.) functions should react. Resource requirements define the resources that can or should be used to implement the software.

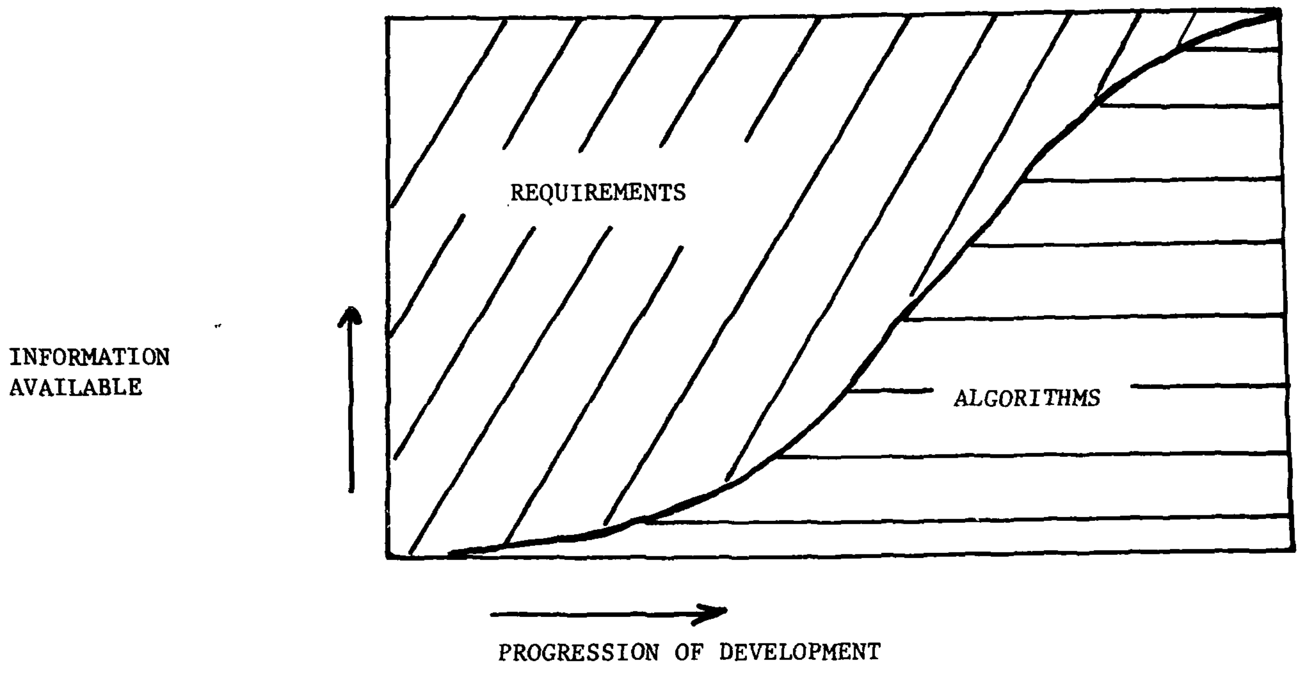


Figure 2. Transformation of Requirements to Algorithms

In the analogy between software development and housing construction, the architectural plan, electrical/plumbing systems, and cost can be compared to the functional, performance, and resource requirements of software. One of the problems in constructing a house is to put the three different and distinct systems together in a way that is both aesthetic and convenient for household activities. Similarly, one of the big problems in software development is to put the three sets of different, distinct, but interrelated requirements together so that the final product satisfies all of them. However, in housing construction, the structure of a house is a dominant part and the electrical and plumbing systems have to comply. The length of the wires and pipes used will generally be considered unimportant. Consider a hypothetical question: How does an architect produce a house plan (all the drawings of a house, including electrical systems) if the requirements are given in terms of X yards of wires, Y feet of pipes, and maximum cost Z dollars, in addition to the regular requirements? Obviously, this creates a more complicated problem than currently practiced. The architect may conclude that either unconventional house structures need to be used or some requirements are not feasible. This hypothetical situation is more nearly the problem that the software professional faces.

Still another parallel can be drawn from the design of the house and the role of software requirements. Given a solution to part of the problem (e.g., floor plan) one can make more intelligent demands on the other solution parts (e.g., wiring and piping). This is the classic ill-structured problem--one for which a symbiotic relation exists among the solution parts [11] . The impact this has on requirements is that it places constraints on what form information may best be stated at what stages of development. Specifically, providing information may best be stated at what stages of development. Specifically, providing information at a level of detail which has no impact on the next increment of algorithm development is as productive as specifying the length of the piping at the outset of design of the house.

SECTION III. A HIERARCHY OF REQUIREMENTS

Figure 3 is an illustration of the software development process. The three vertical edges of the pyramid stand for the functional, performance, and resource requirements of software. The top cross section represents a set of software requirements that encompass all the three major aspects. The software development process can be summarized as a sequence of transformation processes that bring the software requirements to the final product. Each cross section represents an abstracted solution of the problem with a certain set of primitives. Each transformation includes the realization of the higher level primitives by the lower level primitives. This transformation involves the conventional connotation of refinement, design, and implementation. Three basic activities in this process of transformation are: (1) allocated performance and resource requirements from the level η primitives to level (η + 1) primitives; (2) realize the primitives in level η by the primitives in level (η +1); and (3) demonstrate that the abstracted solution in level (η +1) does exactly the same as the abstracted solution in level η and that the allocated resource and performance requirements are feasible and correct. Two points need to be clarified before any meaningful discussion can continue. The first point is that, at any stage, both requirements and algorithms may co-exist in a single document called the software specifications. Most people agree that a software specification is the product of software design written in a specified form. Therefore, a specification is a realization of requirements by a specified set of primitives. Requirements without a partial design that can fulfill them are impractical. A partial design is a flow model or some other equivalent representation that is used to demonstrate feasibility to some extent. For example, one may develop a complete set of requirements for a magic carpet in terms of a complete and consistent list of functions to be performed. Without a baseline design, however, this set of requirements is simply a wish list not to be taken seriously. On the other hand, the baseline design should be used only as a reference -- a starting point subject to feedback. Any future development/transformation should not be bound by it as its main purpose is to demonstrate feasibility. From actual practice, it is natural to write requirements according to some specific design. Thus, the feasibility of the requirements can be kept within known bounds. This is the reason why specifications should thoroughly integrate requirements and algorithm.

The second point is that there should be intermediate levels of specifications from pure requirements to pure algorithms. The number of intermediate levels needed is different from project to project. This represents an opportunity for an alternative measure of software complexity. The complexity of software may be defined by the number of intermediate levels needed to carry out successful transformations. Currently, the software complexity is defined by the number of lines of code in a programming language. Simple programs do exist with

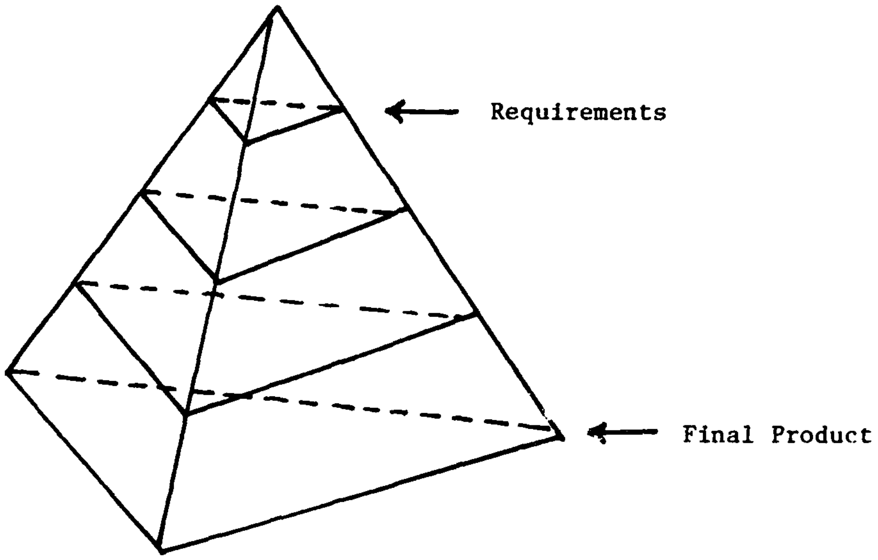


Figure 3. Software Development Process

large numbers of lines of codes, and difficult (complex) programs exist with only a small number of lines of code, for example, the notorious APL one-liners. By using the number of specification levels necessary for the successful completion of transformations, one may find the real meaning of software complexity. At the same time, this proposed concept encompasses all the conventional notions of complexity that current intuition conveys. For example, the assembly language program that implements a design is more complex than its corresponding FORTRAN program, because the former program obviously requires a more detailed specification than the latter to carry out a successful transformation.

Essentially, system design is a mapping process that maps a set of requirements into abstract, feasible solutions. Obviously, each level of solution is closer to implementation than the previous one. This hierarchical approach portrays a sequence of transformations from one level of abstract solution into the next level of abstract solution. Eventually, a development process is needed to transform an abstract solution into a real system. The last process is called implementation.

A widely used illustration for developing software from requirements is depicted in Figure 4 [12]. This figure probably arises from the frequent contractual obligation for a requirement review, preliminary design review, detailed (or critical) design review, and acceptance test. As useful as this model may be in some contexts, it seems to dictate a transformation process that allows only discrete changes in level of intermediate specifications. For any existing, fairly complicated software, no one would deny the use of other levels of specification before the final software specification. Because the other levels are not contract deliverable items, however, no one documents them. The apparent sufficiency of this tri-level specification conceals the necessity of more levels for complex systems. The failure to document those needed intermediate levels of specification can often be pinpointed as the cause of errors/modifications or even project failures. Omitting levels usually means higher completion risks.

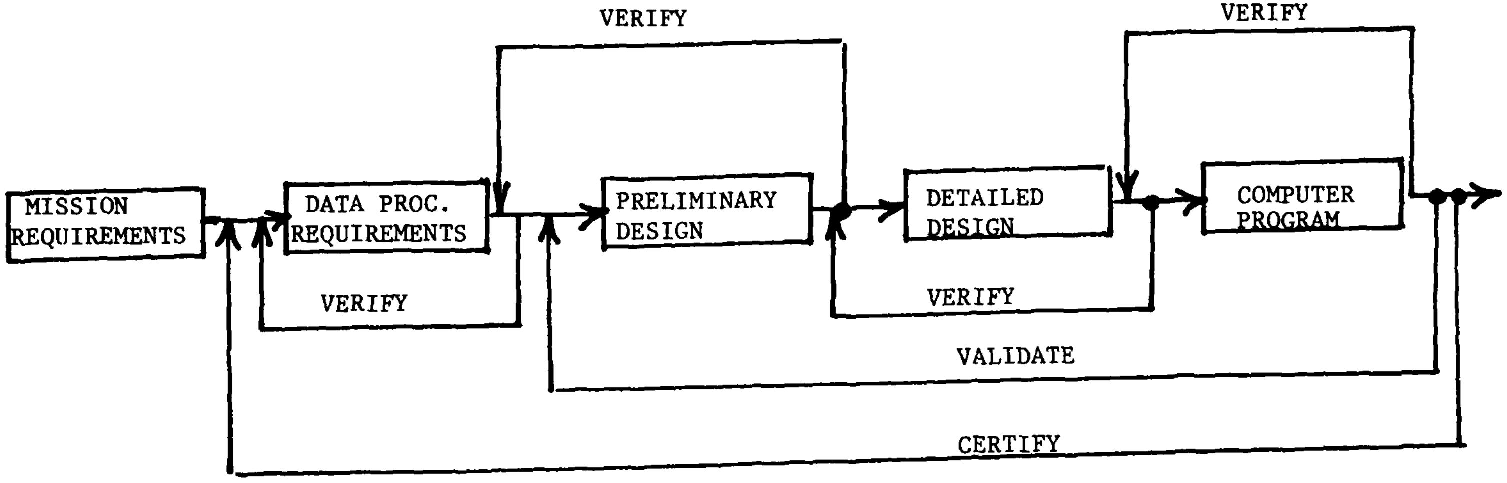


Figure 4. Software Development Cycle

SECTION IV. SOFTWARE REQUIREMENTS PROBLEMS

The major requirements problem has to be identifying how many and of what form should the intermediate levels be. Is there a single answer or even a single approach to an answer? Current wisdom indicates a unified approach to implementation (e.g., higher order languages, structured programming, and levels of abstraction), but indicates there are different paths leading to implementation. Figure 5 indicates why. There seems to be a continuum of applications extending from those that are data dominated and requirements emphasize functional characteristics to those that are control dominated and emphasize performance criteria. Without losing sight of this separation (to which we will return) let's examine some of the unifying characteristics.

For the moment, let's concentrate on how software requirements are derived from the total system requirements. Given an existing or speculative problem that demands some automated processing capabilities in its solution, one must go through many steps to completely develop a system that solves the problem. The steps involved typically are represented as follows:

Requirement Analysis

System Design

Subsystem/Interface Design

Subsystem/Interface Development

System Integration

Checkout.

Requirement analysis is a process that involves mapping inputs to the desired outputs and deriving the functional requirements and their corresponding performance and resource requirements. System design consists of at least four important activities: subsystem identification, function and performance allocations, tradeoff analysis, and feasibility studies. Subsystem/interface design is very similar to system design except that a finer detail is to be achieved. At present, during the system design step, many hardware subsystems are easily identified, and their capability limits can be extrapolated from previous experiences. Portions of the system requirements can be assigned to different hardware subsystems. While the remaining portions of the system requirements are assigned to be performed by the data processing subsystem, the data processing subsystem requirements are then used to produce software r equirements. Unfortunately, in many projects the type of computer hardware to be used had been decided long before the data processing subsystem requirements are derived. Under

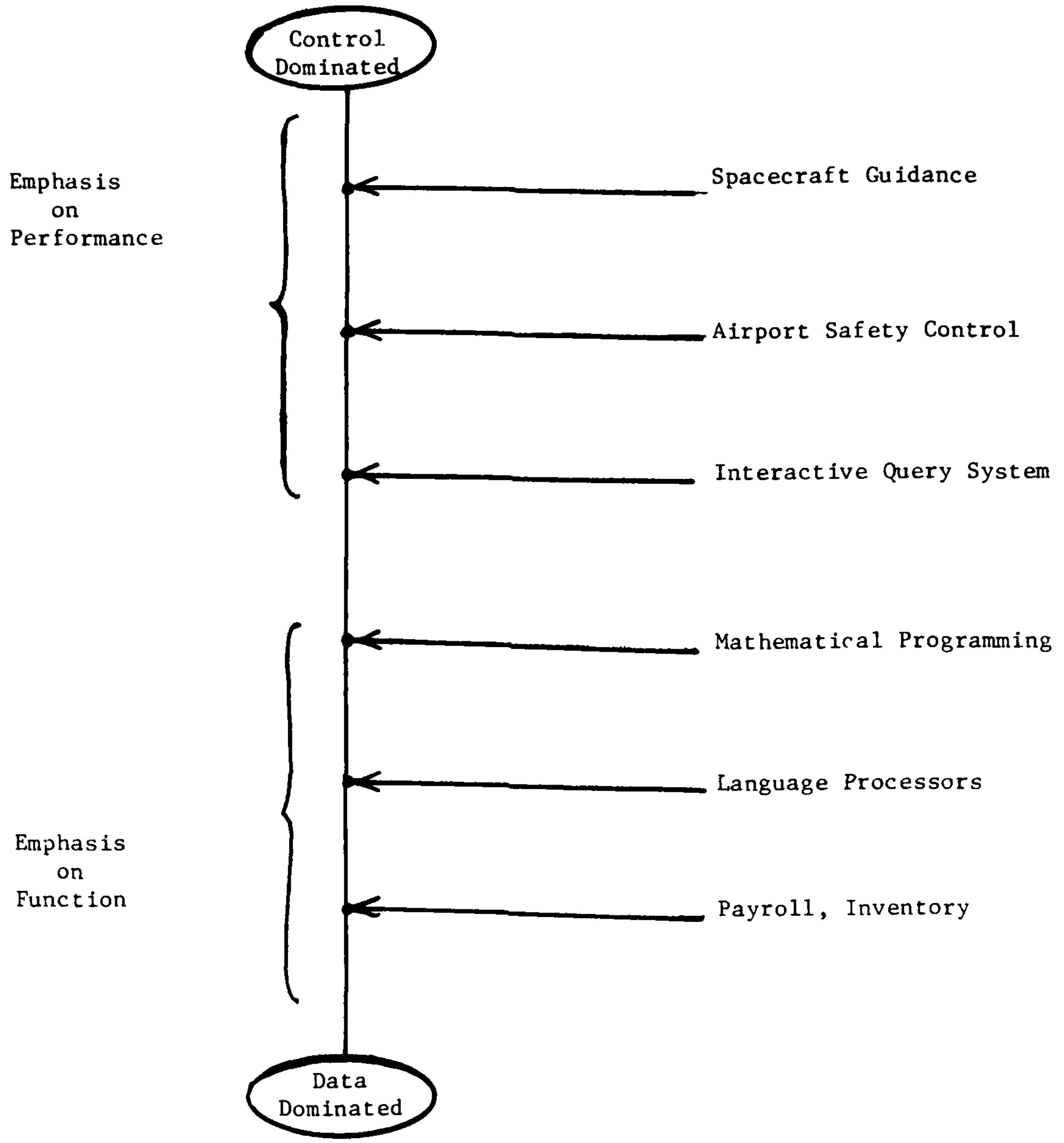


Figure 5. An Application Spectrum

these situations, software requirements are a set of required functions that have to work on the given computer hardware to achieve the data processing subsystem requirements. Because software is less tractable than hardware, frequently it is difficult to determine from the requirements if there exist feasible software solutions. This sometimes causes a considerable amount of trial and error in the system development process.

To illustrate several additional problems, if one considers the base of the pyramid in Figure 3 as an actual solution, more than one pyramid may be associated with a given set of requirements. That is, given one set of scftware requirements, one can derive many different software structures to fulfiii the requirements. The subfunctions should be one step closer to the software code. This process is sometimes called composition. Many levels of functional decomposition may be performed before the software is actually realizable by a special programming language as shown in Figure 4. As mentioned, the complexity of a software system should be measured by the level of decompositions needed. If the decomposition step can be formally defined, then a new definition based on the number of decomposition steps needed will be more reflective of the term "software complexity."

A still more difficult problem is demonstrating that one level of a functional decomposition actually satisfies the performance and resource requirements. How does one allocate performance and resources to subfunctions systematically and optimally? Even after the performance and resource requirements have been allocated to one set of subfunctions in a decomposition, how may it be known that all subfunctions are realizable? At any level there are many different ways of allocating the performance and resource requirements. As the levels of decomposition grow larger, the problem grows more complex.

Another major problem in software requirements is "what constitutes a necessary and sufficient set of software requirements?" Requirement errors have been classified [3] as:

Missing/Incomplete/Inadequate Requirements

Incorrect Requirements

Inconsistent/Incompatible Requirements

Unclear Requirements

New/Changed Requirements.

Without intending to slight consistency analysis, a measure of sufficiency or completeness is more difficult to obtain. The reason is that consistency is based on internal coherency. Thus, the universe of discourse is the requirements document itself. Completeness, however, pertains to the entire problem domain of the application--accepting the relevant and rejecting the irrelevant.

Summarizing, the current problems in software requirements can be grouped into six categories:

1. Software Requirement Statements--How may software requirements be extracted from system requirements?

2. Software Requirement Feasibility Demonstration Technique--How may the feasibility of software requirements be economically demonstrated?

3. Software Functional Requirement Decomposition Technique--How may functional requirements be decomposed into subfunctions?

4. Software Performance Requirement Allocation Technique--How may performance requirements be allocated to subsystems in an optimal way?

5. Software Resource Requirement Allocation Technique--How may resources be systematically allocated to different functions?

6. Specification Completeness--What constitutes a necessary and sufficient set of software requirements?

SECTION V. CURRENT PRACTICES

Existing software systems are produced from given software requirements. This does not negate what has been discussed in the previous section about software requirements problems. It only indicates that there are software requirements that are feasible and that can be used to guide the development of software systems. The majority of the software problems are overcome in those developments by a trial-and-error process. This, as mentioned before, is part of the problem of cost and/or schedule overruns of software projects.

The great majority of current practices in developing software systems from software requirements can be roughly described as follows. First, the software requirement statements are studied by a (group of) highly experienced staff. Then an abstracted functional diagram that may fulfill the set of software requirements is produced. This process is highly dependent on the experiences of the people who perform this task. At this level the problems in requirement statements usually are not fully discovered. This leaves sifnificant problems during the software development process. Some of the problems discovered later may require a totally new start.

After an abstracted functional diagram is produced, some functions in the diagram are allocated with the performance requirement if the functions are detailed enough to allow this analysis. The allocation is done through a combination of the participating personnel's experiences, analysis, and random judgment. This also leaves plenty of room for future corrective effort. Very little attention is paid to the resource allocation at this step. There may be implicit resource allocation consideration during the performance allocation phase.

From this point on, an iterative decomposition process will be performed. Each function in the abstracted functional diagram is divided into subfunctions. A more detailed level of functional diagram is formed, but there does not exist a formal concept of levels of abstract solution. The same process of performance allocation as before is done here. This decomposition will be done repetitively until a level is reached in which each subfunction is implementable by a specific programming language.

At each level of decomposition, the performance and resource allocations are derived according to the participating personnel's experience analysis and judgment. The feasibility of the abstract solution at each level is also considered subjectively but seldom simulated. Whenever a problem is found (i.e., infeasibility of certain subfunctions), a level-by-level trace back of redesign is initiated. The impact of any design change will have to be analyzed and an assessment of the impact studied before continuing to the next level of decomposition. Many serious software requirement errors are discovered in this process. Sometimes the feasibility of the original software requirement is in doubt to the extent that some requirement will be changed to carry out the project. Much wasted manpower and time is incurred because of the latent discovery of errors in software requirements [3, 5] .

Because of trial-and-error practices used in the software development process, there is no systematic way of checking the successful completion of each step during the software development process. Many errors in software requirement statements were not detected until much effort had been wasted to try to satisfy the wrong requirement. Research is needed to identify and state clearly the necessary and sufficient software requirements and the associated processes and techniques that must be used in development.

A summary of the problems in current practice is:

> The technique used in transforming software requirements to the final product is ad hoc and problem-dependent. No systematic approach is employed to solve the general problem of the transformation process.
>
> The steps and levels needed in the transformation are not well defined. This vagueness contributes many problems for any potential modification.
>
> Too many new start possibilities are in the existing development process. This makes an accurate estimation of eventual project effort most difficult.

SECTION VI. RESEARCH ISSUES

Many experts are initiating important research to correct these problems [6, 7] . This section provides a summary of existing problems in the software requirement area, with justification and rationales for each.

1. What information is necessary and sufficient in software requirements?

   Currently, software requirements are voluminous for large projects. It takes much time to read through them, not to mention analyzing them for errors. If software requirements can be characterized by a list of items, one can simply check them individually and apply the divide-and-conquer technique. One can definitely use this characterization to write better software requirements. Also, the characterization can be used to evaluate given software requirements.

2. What are the characteristics of software requirements and how to check them?

   Many professionals have tried to determine the desired characteristics for software and requirement statements. Completeness, consistency, feasibility, testability, etc. are all included. Nevertheless, these characteristics, if defined according to the natural meaning of the terms, are very difficult to check. Therefore, precise definitions of these desired characteristics should be given and research conducted in problem detection.

3. How may a hypothesis on software requirements of large-scale systems be tested without actually performing a similar scale project?

   Brooks [8] and Weinberg [9] noted that it is difficult and unrealistic to try to extrapolate conclusions drawn from small-scale projects and try to apply them to large-scale projects. At present no sponsoring agent would support large-scale experiments on software hypothesis. Without any reasonable substitution, one will either perform a small-scale project and try to extrapolate the results to a large-scale software or implement a real project using an unfounded hypothesis. The effort proposed and practiced at Toronto by Horning [7] belongs to the former category.

4. How may software performance requirements be allocated to the next level functions?

   It is known from the model of software development process that every function in level i will have to be refined to a group of subfunctions in level (i + 1). At the same time, the performance requirements set for the function in level i should be optimally distributed to the subfunctions in level (i + 1). Presently, this is done by personal experience and subjective reasoning until errors are found. Trial-and-error is by no means a scientific process. Clearly the performance allocation problem is not simply a linear programming problem.

5. How may software resource requirements be allocated to the next level functions?

   Because resource allocation problems may directly affect the performance of a function, it is crucial that this problem be solved before any true advances can be achieved in large-scale software development. A typical example to show that the resource allocation directly affects the performance is an information retrieval system. If such a system is to be developed and the response to any retrieval command is very stringent, then the use of core, disc, drum, or tape for certain functions will play a crucial role in the success of the eventual system.

6. How may it be demonstrated that (1) the functions in level (i + 1) actually realize the functions in level i and that the performance and resource requirements allocated to the level (i + 1) functions are properly derived from those requirements set for the functions in level i; (2) the functions in level (i + 1) with their respective performance and resource requirements are potential candidates for hardware?

   In the hierarchical software development process, the specification in level i should be transformed and realized by the specification in level (i +1). To ensure that the transformation is valid and that the specification in level (i + 1) is feasible, one has to find ways to answer problems 1 and 2, respectively. The entire development process depends on the satisfactory answers to these problems. Until a satisfactory answer is obtained, software development will remain a totally individual-experience-dependent, trial-and-error process.

SECTION VII. COMMON FEATURES IN AUTOMATED REQUIREMENTS SYSTEMS

All the existive automated requirements systems, including SADT (SofTech), SREM, (TRW), DAS (Hughes), and IORL (Teledyne Brown Engineering), are trying to rectify the present software development problems. All of the requirements systems have claimed that usage of their systems will result in savings of software cost and attaining better quality software. We would like to point out that because of the ad hoc and the trial-and-error nature of the existing practice in analyzing software requirements, it is not a surprise that the newly developed automated tools work as this. This is by no means to slight the effort and goals of the software requirements systems. On the contrary, we believe the achievements in software requirements have been promoted by these automated systems. It is due to these automated systems that we have shown that software can be produced less expensively and with higher quality.

After studying the existing software requirements systems/ methodologies, a list of summary of the common features of these systems were derived. They are described and explained below.

(1) Graphic techniques are used to represent system structures to facilitate human analysis and communication. It has been pointed out repeatedly that the main problem in large-scale software development is communications [8]. Due to the sheer number of people involved in a project, the chance for one to misinterpret another person's work is tremendous. With the aid of graphically representing a structure in addition to linear text description can help eliminate a majority of the misinterpretation problems.

(2) Hierarchical structure is used in describing requirements. This is due to the fact that in order to arrange a large amount of data into an easily understandable form, hierarchical structure is the only way we know how to do it efficiently.

(3) Modular approach is adopted to describe requirements. No one can sensibly analyze a large textual description without losing some finer details during the analysis. Our abstraction capability is, therefore, again adopted in this problem and modularization of requirements becomes a dominant feature in requirements analysis. It is also caused by the graphic representation of structures of requirements, since without modularization no output of graphic representation may be feasible for comprehension.

(4) More effort is demanded in analyzing and describing requirements before actual design and implementation of a project can begin. This is because of the rigor needed in input, the requirements in an appropriate language and review of the output textual analysis of the requirements (and sometimes dynamic simulation of the requirements). The significance of all these methodologies/techniques seems to point out that more effort spent in the requirement analysis phase will provide greater benefit in the entire life cycle of the software. However, at the current time, no one can determine the optimal allocation of effort in the requirements phase. This is our next research problem, viz. deciding the proper allocation of resource/effort in a software development model to produce the best product.

REFERENCES

1. Boehm, B. W., "The High Cost of Software," Practical Strategies for Developing Large Software Systems, ed. Ellis Horowitz, Addison-Wesley 1975.

2. Hopcroft, J. E. and Ullman, J. D., Formal Languages and their Relation to Automata Theory, Addison-Wesley, 1969.

3. Bell, T. and Thayer, T., "Software Requirements: Are They Really A Problem?" Proceedings 2nd International Software Engineering Conference, October 1976.

4. Schwartz, J. I., "Construction of Software: Problems and Practicalities," Practical Strategies for Developing Large Software Systems, ed. Ellis Horowitz, Addison-Wesley, 1975.

5. Meseke, D. W., "Safeguard Data Processing System: The Data Processing System Performance Requirements in Retrospect," Bell System Technical Journal, Special Supplement, 1975.

6. Boehm, B. W., "Experiments on Software Specifications & Design," Private Communication.

7. Horning, J. J. and Wortman, D. B., "Software Hut, A Computer Program Engineering Project in the Form of a Game," to be published in July 1977 issue to IEEE Transactions of Software Engineering.

8. Brooks, F. P. Jr., "The Mythical Man-Month," University of North Carolina, 1975.

9. Weinberg, G. M., "The Psychology of Computer Programming," Van Nostrand-Reinhold, 1971.

10. Hamilton, M. and Zeldin, S., "Higher Order Software--A Methodology for Defining Software," IEEE Transactions on Software Engineering, SE-2, 1 (March 1976), 9-32.

11. Simon, H. A., "The Structure of Ill Structured Problems," Artificial Intelligence, 4,3 (Winter 1973), 181-202.

12. Reifer, D. J., "Automated Aids for Reliable Software," Proc. Intern. Conf. on Reliable Software, April 1975, 131-142.

USER EXPERIENCE WITH A FORMALLY DEFINED REQUIREMENTS LANGUAGE IORL (INPUT/CUTPUT REQUIREMENTS LANGUAGE)

Charles R. Everhart
Teledyne Brown Engineering
Huntsville, Alabama

ABSTRACT

A formally defined requirements language (i.e., complete syntax rules and semantics) with the appropriate characteristics is necessary for precise system definition. In addition, a disciplined application of this language during the system requirements definition phase can provide information necessary (and usually not available) for the subsequent phases of system development. This paper describes such a language (IORL) which has been used in both system requirements definition and design phases. The description includes a brief example of the language syntax and semantics, as well as a description of an interactive graphics system used to store, modify, and retrieve IORL symbols and tables.

Finally, user experience with and response to IORL is discussed in terms of its success in controlling a project effort relative to the rigor with which IORL is applied.

USER EXPERIENCE WITH A FORMALLY DEFINED REQUIREMENTS LANGUAGE IORL (INPUT/OUTPUT REQUIREMENTS LANGUAGE)

Charles R. Everhart
Teledyne Brown Engineering
Huntsville, Alabama

ABSTRACT

A formally defined requirements language (i.e., complete syntax rules and semantics) with the appropriate characteristics is necessary for precise system definition. In addition, a disciplined application of this language during the system requirements definition phase can provide information necessary (and usually not available) for the subsequent phases of system development. This paper describes such a language (IORL) which has been used in both system requirements definition and design phases. The description includes a brief example of the language syntax and semantics, as well as a description of an interactive graphics system used to store, modify, and retrieve IORL symbols and tables.

Finally, user experience with and response to IORL is discussed in terms of its success in controlling a project effort relative to the rigor with which IORL is applied.

INTRODUCTION

A number of recent articles have indicated that the costs of developing software will represent 50 to 90 percent of the total costs of data processing systems by the mid 1980's [4,6]. In spite of the proliferation of programming languages [1] and software development techniques devised during the past 20 years, the industry has been unable to change this upward trend in software costs. Software development productivity has actually decreased over the last 10 years according to one source [3].

The reason new programming languages and techniques may have been ineffective in reducing software costs is that they solve the programming problem but do nothing for the "System Requirements Definition" problem. The significance of this fact is illustrated by a recently developed large software project. It was reported that only 17% of the cost of this 7.5 year development effort was associated with software coding and unit testing [6]. The major cost (83%) was in system definition, system design, system testing and system integration activities. The last three (design, testing, and integration) are very much dependent on the quality of the first (system definition). That is, system definition (if properly applied) can provide: the requirements for design, the criteria for testing and the structural plan for integration.

Therefore, it would appear that what has been commonly referred to as a "Software Development" problem is in reality a "System Definition" problem. If the industry expects to see large reductions in software development costs, it must honestly attack the system definition problem.

The overall motivation behind the development of IORL is to provide a precise, comprehensive, yet convenient language for describing system definition and design requirements. In order to effectively deal with the subtle (as well as obvious) problems of a large system, it was assumed that the language should have the following characteristics:

- The language must consist of quantitative, scientific/engineering notation controlled by a set of formal syntax and semantics.
- It must allow the comprehensive description of a total system including hardware, software, and manual functions.
- It must be a unified language which allows both system definition and design requirements to be specified within a hierarchical structure.
- It must be modular in that precise communication of information at one level in the requirements documentation structure does not require understanding of information contained in levels above and/or below.
- The language must provide visibility through clearly defined placement of requirements information independent of the system type to be described.
- It must provide requirements traceability to determine where a particular requirement was originated and/or implemented.
- It must provide a hierarchical structure which ensures the ability to resolve requirements and/or design conflicts and responsibilities.
- The language should be basically graphic in order to take advantage of the available interactive computer graphic systems as well as providing a more natural interface for the system analyst.

Teledyne Brown Engineering has been actively involved in this type of development since 1971 and has devised a requirements language called IORL (Input/Output Requirements Language) which claims the objectives of the preceding discussion.

IORL

IORL is a formally defined language (syntactically and semantically) [2,5] which uses a combination of both graphic symbols and mathematical notation to express system definition and design ideas. Block diagrams (analogous to those used in control theory) organized in a hierarchical manner identify the parts of a system and the interfaces between these parts at all levels of system definition and design. Descriptions of each interface identified are contained in a set of tables called IOPTs (Input/Output Parameter Tables). Another diagram called an "IORTD" (Input/Output Relationships and Timing Diagram) is used to define the total transformation function from input to output as well as the response time requirements for each and every block in the hierarchy. These diagrams (analogous to a "Transfer Function" in control theory) provide the symbols for specifying the sequential, simultaneous, logical, mathematical and timing requirements between inputs and outputs of each given block. The elements of IORL and hierarchical structure of requirements information are characterized in Figures 1 and 2.

IORL STORAGE AND RETRIEVAL FACILITY

Storage, retrieval and modification of IORL diagrams and tables have been implemented on a standalone PDP/11 based graphics terminal (GT44 and GT46) with 16K memory. The interactive graphics system includes a 17 inch refresh type graphic screen with lightpen capability as well as an electrostatic printer plotter, which produces 8.5 x 11 inch copies of the screen. In edit mode, IORL information is entered by pointing the lightpen at a location on the screen and then pressing the keyboard button associated with the desired symbol. In display mode, system details are accessed by directing the lightpen to points of interest on the higher level diagrams of the hierarchy. This results in the subsequent display of these details on the screen. Requirements diagrams are stored on and retrieved from disk packs which are also a part of the facility.

BMD (BALLISTIC MISSILE DEFENSE) PARTITIONING STUDY EXAMPLE

Not only is a formal requirements language necessary in specifying precisely the functional and performance characteristics of the system at all levels of definition and design, but at the same time this information can be used to predict the costs in time and money required to develop, implement, operate, and maintain a proposed system. Other benefits derived from this information include the direct generation of system and environment models used in the analysis of design solutions, direct generation of test criteria to be used during the test and integration phases of system development and the providing of a vehicle for maintaining configuration control throughout the life cycle of the system.

In an attempt to determine the feasibility of the preceding thesis, a study (entitled: "BMD Partitioning Study") was performed. The

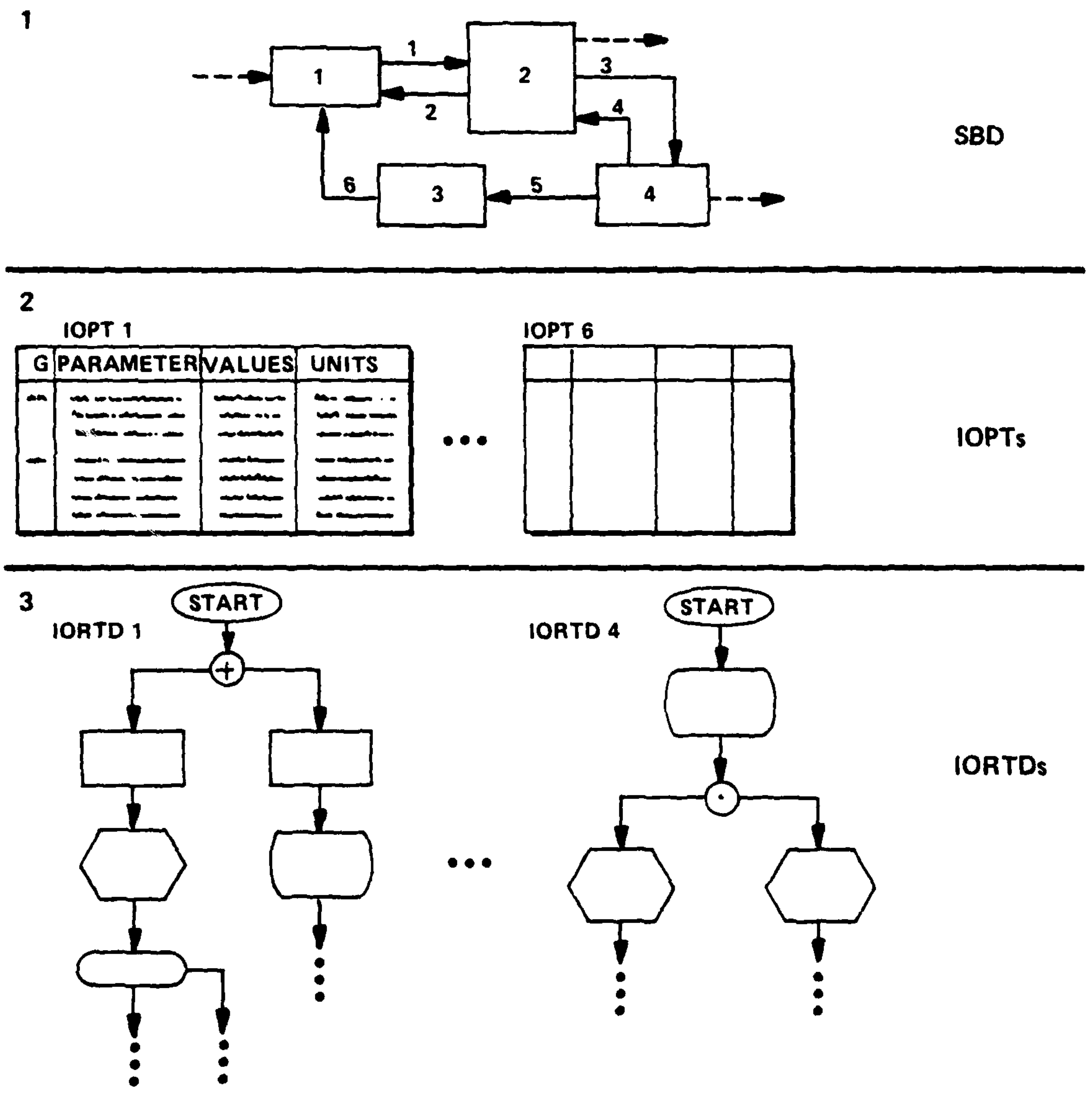


FIGURE 1. IORL ELEMENTS

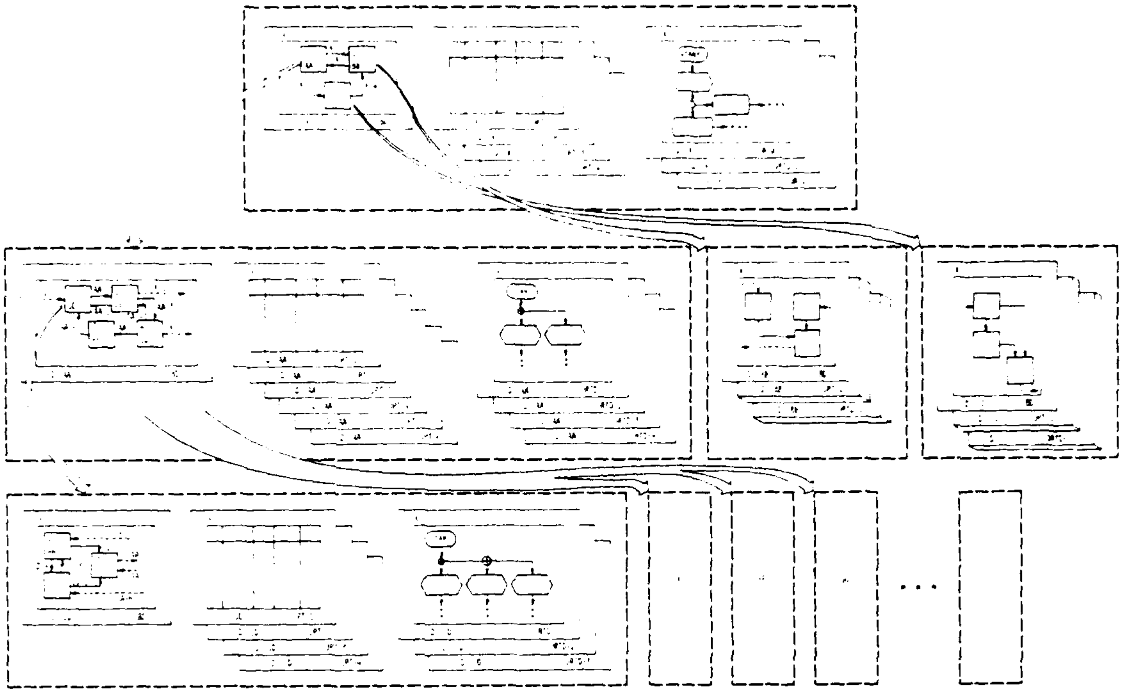


FIGURE 2. IORL INFORMATION HIERARCHY DIAGRAM

objective of this study was to demonstrate that certain quantitative characteristics, related to system development and operation costs, could be derived directly and MECHANICALLY from formally defined system requirements specifications. IORL was used as the language for specifying the definition and design requirements.

The demonstration consisted of four basic steps. First, the BMD system requirements and its environment were defined using the complete set of IORL symbols, tables and diagrams. This information, which represented the first level in the system hierarchy, was the ONLY information used in the subsequent requirements analysis and design activities. After checking the first level specification for completeness and consistency, the second step involved designing two different solutions to the BMD requirements, again expressing these solutions in IORL and placing this information in the second level of the system hierarchy. The purpose in specifying two design solutions was to compare the total system costs which resulted from each of the solutions. In the third step, each completely specified solution was validated against the BMD requirements by exercising the environment, BMD, and solution models (all written in IORL) and then comparing responses at the BMD/environment interfaces. In this manner it was established that each solution responded to the environment exactly as required by the BMD requirements (model). If not, the design solution was corrected. In the final step, each solution was evaluated to determine its effect on total system costs. This evaluation, which was a combination of static and dynamic analysis of only that information contained in the IORL specifications, produced the summary partition evaluation results shown in Table 1. These summary results were derived from a series of intermediate results which plotted for example bandwidth for each interface as a function of time, storage required as a function of time, functional speed required, etc.

Our experience with this study has resulted in the following conclusions:

- Quantitative measures related to the costs of a system can be determined from an analysis of system definition and design requirements.
- The requirements language used to specify system definition and design facts is the key factor in the success of the preceding demonstration (i.e., the language must have certain characteristics and enforce certain disciplines).
- The proper specification of definitior and design requirements can provide information necessary to all phases of a system development (definition, design, implementation, test and integration).

TABLE 1. PARTITION EVALUATION RESULTS

	PARTITION 1				PARTITION 2			
	GROUND	TRACKER	SENSOR	TOTAL OR WORST	GROUND SYSTEM	STTCOR	SENSOR	TOTAL OR WORST
RELIABILITY	0.975	0.900	0.975	0.941	0.980	0.980	0.980	0.941
STORAGE (kbits)	23.5	389.3 (2)	2.67	804.8	138.6	284.4	27.9	432.7
PROCESS SPEED (η sec)	5.44	5.85	5.56	5.44	0.814	0.814	0.814	0.814
PROGRAMMING COMPLEXITY (MAN-MONTHS)				35.65				26.3
PEAK BANDWIDTH (Mbits/sec)				2.52				20.94

USER EXPERIENCE SUMMARY AND CONCLUSIONS

Comments on IORL user experience should be accepted in light of the following background. Approximately 45 analysts have used IORL both manually and with computer support during the past three years. No formal instruction on the proper application of the IORL technique has been given to these users. Their instruction has consisted of reading three manuals (77 total pages) which define IORL syntax rules and semantics. No formal study has been performed to determine user experience and in most cases the user comments were qualitative and quite subjective.

The IORL technique has been applied to a number of different systems, including weapon systems (BCS, HWS, TSQ-73, PATRIOT, MINISD, MSMX1, etc.), development of simulation programs, analysis of management plans, development of interactive graphics systems, the development of a remote batch entry system (UT200), MIS project and others. The approximate size of each of these projects ranges anywhere from 50 pages or IORL information to several thousand pages. Currently, the largest system is about 3800 pages.

IORL has been used as an analysis tool for determining faults within existing and/or proposed systems. It has been used to obtain control of poorly or undocumented existing systems (this usually results in the uncovering of previously undetected discrepancies). It has been applied to the total development of several systems (from initial conception to testing the final product). IORL has been used to describe new and/or existing systems from the top down as well as existing systems from the bottom up.

In general, the degree to which IORL syntax rules are followed determines project success. However, when some of the rules are violated, attempts to follow most of the rules usually result in the detection of inconsistencies, ambiguities and incompleteness not otherwise observed. The explicit nature of the language tends to force vigorous pursuit of unanswered questions. Discrepancies and unspecified information is quite noticable. This is exemplified by the following excerpt from a report describing the use of IORL on the analysis of a software development management plan [7].

> "The aspect of the structured analysis that exposed the problems were the additional careful scrutinies and analyses of the SWDMP required in order to develop the IORL graphical representation of the plan."

Other comments of a qualitative nature which were made by users are: "Invaluable in control of large projects, especially during evolution", "Very valuable in highlighting inconsistencies, incompleteness", "(The) finished product and the ease of creation/modification of same outmode the difficult typist/artist/draftsman approach to documentation."

The IORL facility has been in operation almost 20 months. The fact that during this period, 23 different systems have been entered consisting of approximately 6000 formally accounted (charged) IORL pages, is the best indication of user response to IORL and the facility. A portion of this usage can be attributed to appeal of the computerized IORL storage and retrieval facility. However, IORL has been used successfully prior to the facilities existence. These prior applications include the development of a "Ballistic Trajectory Simulation" program, the development of a "Interactive Missile Design Tool", the development of the IORL facility itself, and a 650 page requirements description of the BCS (Battery Computer System). Table 2 indicates user success with IORL relative to the first three systems just mentioned. Although productivity (in terms of "Lines of Source Code Per Hour") is not the best measure, it provides an order of magnitude in determining the success of these projects relative to the application of IORL. Comparison of the first line in Table 1 with the last three indicates an increase of approximately double productivity when the IORL facility is available.

TABLE 2. TBE EXPERIENCE

JOB	LANGUAGE	LINES OF CODE	IORL FACILITY AVAILABLE	SYSTEM EXPERIENCE	PRODUCTIVITY
200 UT	ASSEMBLY	8,650	YES	NONE	8.5 LINES/hr
GRAPHICS FACILITY	ASSEMBLY	8,200	NO	NONE	3.4 LINES/hr
BALLISTIC TRAJECTORY SIMULATION	FORTRAN	2,000	NO	MODERATE	6.5 LINES/hr
INTERACTIVE MISSILE DESIGN TOOL	FORTRAN	39,500 NEW 6,000 OLD	NO	MODERATE	3 TO 3.5 LINES/hr

CONCLUSIONS

From the above information, it is difficult to draw any strong conclusions concerning the effects of IORL on each of the applications individually. This is due to the subjective nature of user comments, the lack of quantitative data generated under controlled conditions, the incomplete state of many of the application systems and in general the newness of the current IORL technique. Where informal application of the IORL syntax rules has been used, evaluation of IORL usefulness must be based on subjective comments of the project managers. Where rigorous application of the IORL rules has been enforced, project success has been easier to measure and more pronounced. In either case, IORL users expressed satisfaction in general with IORL over previous methods.

REFERENCES

1. Shea, William E., "DOD-1, A Common Language", ISRAD in Touch, Volume 2, No. 1, February 1978.

2. Everhart, C. R., "IORL Analysts Users' Manual", Section 1 - Syntax, Teledyne Brown Engineering, May 1977.

3. Dolotta, T. A., et al, "Data Processing in 1980-1985", John Wiley and Sons, 1976.

4. Boehm, B. W., "Software and Its Impact: A Quantitative Assessment", Datamation, pp.48-59, May 1973.

5. Everhart, C. R., "IORL Analysts Users' Manual", Section 2 - Semantics, Teledyne Brown Engineering, May 1977.

6. Ramamoorthy, C. V., et al., "Software Requirements and Specifications: Status and Perspectives", Engineering Research Recommendations, Draft Appendix A, August 12, 1977.

7. Polan, Marvin and Sansone, F. J., "Software Design Methodology Interim Report", Technical Report SD78-BMDSCOM-2211, prepared for Systems Technology Project Office, U.S. Army Ballistic Missile Defense Systems Command (Contract Number DASG60-78-C-0002), May 1978.

Software Requirements Engineering Methodology (SREM)

M. W. Alford

TRW Defense and Space Systems Group

The Software Requirements Engineering Methodology (SREM) was developed for the Ballistic Missile Defense Advanced Technology Center (BMDATC) to address the generation and validation of Software Requirements for Ballistic Missile Defense Weapons System. At that time, REVS was operational only on the Texas Instruments Advanced Scientific Computer (TI ASC), and the methodology had been applied to one moderate-sized "proof of principle" demonstration.

SOFTWARE REQUIREMENTS ENGINEERING METHODOLOGY (SREM) CURRENT (1978) STATUS

M. W. Alford

TRW Defense and Space Systems Group
Huntsville, Alabama

1.0 INTRODUCTION

The Software Requirer nts Engineering Methodology (SREM) was presented two years ago at the Second International Software Engineering Conference [1]; the SREM support software, the Requirements Engineering and Validation System (REVS), was also presented then [2]. SREM was developed for the Ballistic Missile Defense Advanced Technology Center (BMDATC) to address the generation and validation of software requirements for Ballistic Missile Defense Weapons Systems -- the motivation and environment for this research has been previously described [3]. At that time, REVS was operational only on the Texas Instruments Advanced Scientific Computer (TI ASC), and the methodology had been applied to one moderate sized "proof of principle" demonstration problem. The research results were presented at ISRAD in Arlington, Virginia on 23 and 24 May, 1977.

Since then, SREM has been successfully applied to both the generation and independent validation of software requirements for several systems. REVS has been transported to a number of other host computers, and its performance has been improved. The methodology has been successfully transferred to a number of other organizations, and applied to a wider class of problems. The purpose of this paper is to provide a status report of the SREM requirements development procedures, requirements language, support software, and transfer of this technology to other organizations, and to provide an overview of plans for extensions and improvements.

SREM is based on a Graph Model of Software Requirements [4] which is an extension of the Graph Model of Computation [5]. The basic concept underlying SREM is that design-free functional software requirements should specify the required processing in terms of all possible responses (and the conditions for each type of response) to each input message across each interface. Thus, functional requirements identify the required stimulus/response relationships, and autonomously generated output. These required actions of the software are expressible in terms of Requirements Networks (or R-Nets) of processing steps. Each processing step is defined in terms of input data, output data, and the associated transformation. Figure 1 presents an R-Net for a Hospital Patient Monitoring System [6] which accepts a measurement of the blood pressure, temperature, skin resistance, etc., for a patient, tests it for validity, records it, requests the next measurement, and tests the measurement against a pre-specified set of upper and lower limits. Note that five paths of processing are identified, which combine into three

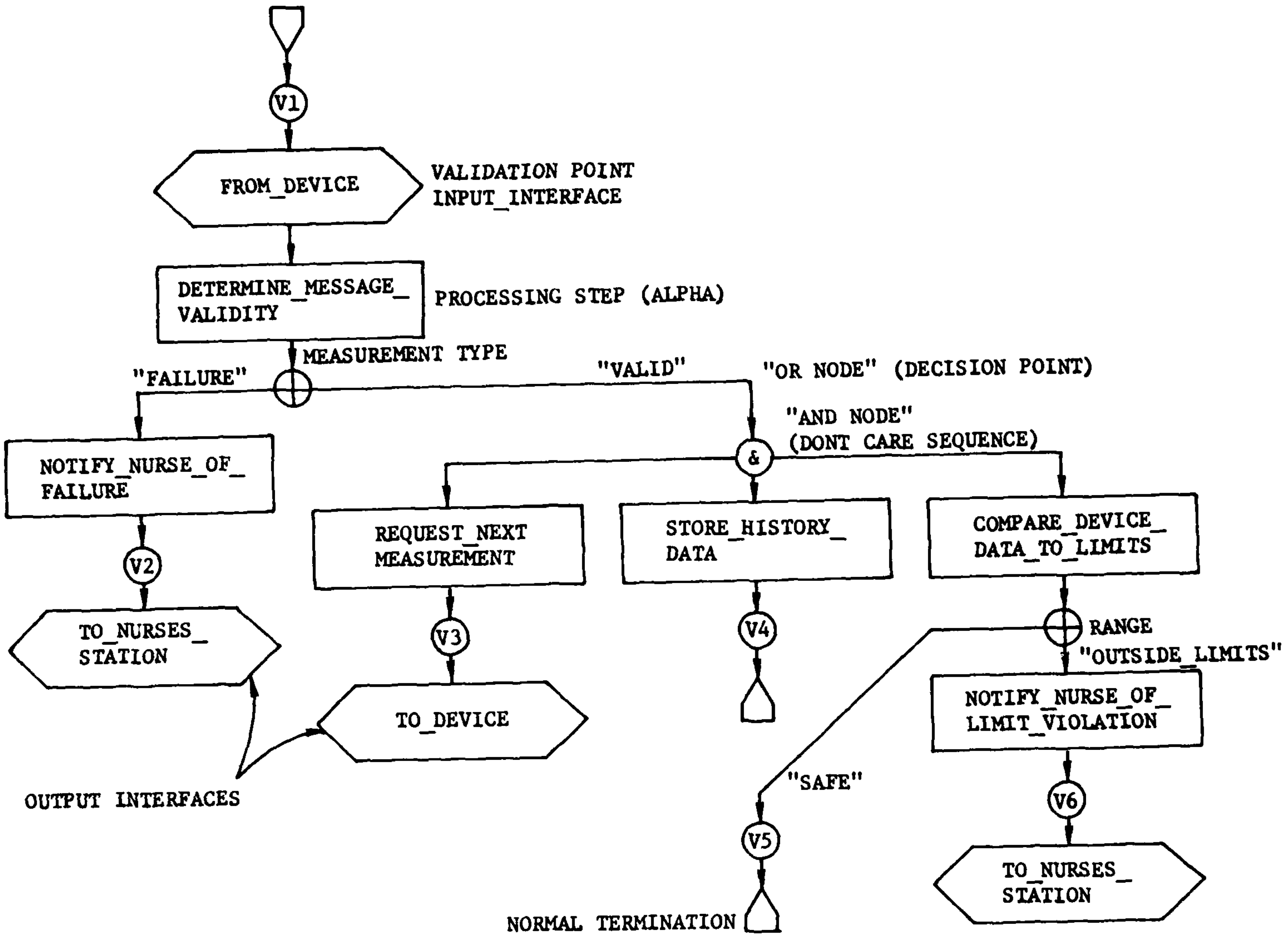


Figure 1 Example R-Net

possible stimulus/response requirements -- the paths to request the next measurement and record the current measurement occur regardless of whether the measurement violates the constraints.

These concepts are embodied in the Requirements Statement Language (RSL). RSL is composed of 21 types of elements, 21 types of attributes, 23 types of relationships, and three types of structures. It is the structures (R-Nets, Subnets, and Paths) and their formal mathematical foundations, and the stimulus-response orientation which distinguish RSL from the traditional techniques for stating software requirements (e.g., the PSL [7] approach, or standard DoD Military Specifications [8]). Table 1 presents a subset of the requirements for the Patient Monitoring System expressed in RSL.

REVS is a large software tool that handles a potentially large data base of requirements, therefore requiring a host computer with a large effective memory space and a moderately fast instruction rate.

REVS accepts RSL as input, translates it into an automated requirements data base, and provides a set of capabilities for analyzing and manipulating this data base. Specific capabilities include the following:

- Translation of an RSL expression of requirements into a central requirements data base.
- Extraction, under user control, of information from the requirements data base for analysis and documentation.
- Identification, under user control, of subsets of the data base for automatic consistency, completeness, and traceability analyses.
- Automated checking of the requirements data base for specific properties of data flow consistency (made possible only because of the underlying formal foundations).
- Graphical representation of the requirements structures, both on-line and off-line.
- Automated generation and execution of simulations directly traceable to the requirements definition.

The REVS program itself consists of over 40,000 executable PASCAL statements, making it the largest PASCAL program known to us. (In comparison, the PASCAL compiler consists of approximately 6,000 PASCAL statements.) An additional 10,000 FORTRAN statements perform data base management functions. The RSL Translator is produced from the Backus-Normal Form (BNF) definition of RSL using the Lecarme-Bochman Compiler Writing System from the University of Montreal, plus additional code for the definition of the semantic actions.

At the time of our presentations two years ago, SREM, RSL, and REVS had been applied to one moderately sized BMD problem to demonstrate capa-

Table 1 Example RSL Definitions

```
MESSAGE:  DEVICE_REPORT.

    PASSED THROUGH:  INPUT_INTERFACE FROM_DEVICE.

    MADE BY:  DATA DEVICE_NUMBER
              DATA TYPE_MESSAGE
              DATA DEVICE_DATA.

DATA:  DEVICE_DATA.

    INCLUDES: DATA PULSE
              DATA TEMPERATURE
              DATA BLOOD_PRESSURE
              DATA SKIN_RESISTANCE.

FILE:  FACTOR_HISTORY.

    CONTAINS: DATA MEASUREMENT_TIME
              DATA HPULSE
              DATA HTEMPERATURE
              DATA HBLOOD_PRESSURE
              DATA HSKIN_RESISTANCE.

    TRACED FROM:  SENTENCE_2
                  SENTENCE_3.

    ASSOCIATED WITH:  ENTITY PATIENT.

ALPHA:  STORE_HISTORY_DATA.

    INPUTS:  DEVICE_DATA.

    OUTPUTS: FACTOR_HISTORY.

    DESCRIPTION:  "THE DATA PROCESSOR SHALL RECORD EACH VALID
                  MEASUREMENT FOR EACH PATIENT".

VALIDATION_PATH:  MEASUREMENT_OUT_OF_LIMITS.

    PATH:  VALIDATION_POINT V1, VALIDATION_POINT V6.

MAXIMUM TIME:  1.

    UNITS:  SECONDS.
```

bility and to illustrate the sequence of steps and associated outputs of the methodology. At that time, it was concluded that the research objectives had been achieved, but that SREM's applicability to other environments and utility in realistic software development environments had yet to be determined.

Section 2.0 contains a summary of the status of REVS installations and the transfer of the SREM technology to other organizations. Section 3.0 contains an overview of the results of several diverse SREM applications. Section 4.0 contains an overview of planned extensions to SREM and REVS. Section 5.0 contains some conclusions of our experiences to date.

2.0 WHERE WE ARE TODAY

The availability of SREM technology for use on software requirements development projects is dependent on two factors: the availability of people who are knowledgeable in the SREM concepts, techniques, and procedures; and the availability of REVS to support the use of SREM. The substantial progress which has been made in both of these areas in the past two years is discussed below.

2.1 TECHNOLOGY TRANSFER

Technology development is fruitless without the mechanism for its transfer to others working in the field. Mere existence of tools, or codification of experience, does not necessarily lead to a transfer of working knowledge which can be exercised by others in operational environments. The true test of a methodology is whether it can be absorbed and applied by others.

To successfully transfer SREM technology to others, three techniques have been used: transfer of documentation of the SREM procedures, language, and software capabilities; presentation of a "short course" in Requirements Engineering; and on-the-job training. Table 2 presents a list of the organizations which are applying these techniques. More details of the transfer are provided below.

Some individuals at the University of California at Berkeley, and at TRW, relied only on the published documentation and achieved mixed results in later applications. Requirements engineers from Johns Hopkins University/Applied Physics Laboratory (APL) and TRW have been provided with the basic documentation and with some on-the-job training consisting of assistance in SREM on a problem of their choice. The on-the-job training provided an intensive information transfer, and the opportunity to correct any misconceptions about the software requirements engineering process and misunderstandings of the language concepts. For example, requirements engineers with previous experience writing software requirements usually have an overwhelming urge to try to define a queueing scheme for buffering input messages in RSL; elimination of this design leads to statements of response time requirements which allows software designers to decide which buffering scheme meets such requirements.

Table 2 Technology Transfer

ORGANIZATION	TRANSFER METHOD			APPLICATIONS		
	DOCUMENTATION ONLY	ON-THE-JOB TRAINING	SHORT COURSE	EXPERIMENTAL USAGE	VALIDATION	SPECIFICATION GENERATION
APPLIED PHYSICS LABORATORY		X		X	X	X
HUGHES			X	X		
McDONNELL DOUGLAS ASTRONAUTICS CORP.			X	X		X
RCA			X	X		X
TELEDYNE BROWN ENGINEERING			X	X		
TRW	X		X	X	X	X
UNIVERSITY OF CALIFORNIA, BERKELEY	X			X		X
U.S. ARMY COMPUTER SYSTEMS COMMAND			X	X		

In November 1977, a three-week course was held at McDonnell Douglas Astronautics Corporation (MDAC) in Huntington Beach, Cal., sponsored by BMDATC, to assist the technology transfer to a wider audience. Participants included requirements engineers from MDAC, Hughes, RCA, Teledyne Brown Engineering, and the U. S. Army Computer Systems Command. Course notes were provided to detail the methodology and to provide examples of inputs and outputs of each methodology step. All participants had the opportunity to define functional requirements for two example problems: a common example for the whole class, and then independent team projects. The course was evaluated by all as successful. Table 2 indicates partial results of this transfer: APL, MDAC, RCA, and TRW all have performed, or are performing, demonstration and/or operational projects for generating and/or validating software requirements using the technology.

Although continual on-the-job training is the ideal, the training course approach was found to be an effective and cost-effective mechanism for technology transfer. Several future courses are under consideration.

2.2 REVS AVAILABILITY

A critical element in the availability of SREM is the availability of REVS to support its application. Two years ago all REVS capabilities were operational, but REVS was available only on the TI ASC in Huntsville, Alabama, which had no remote access capability. This severely limited the scope of its application. Since then, REVS has been installed in several more locations on both TI ASC and CDC host computers with remote access capability. Table 3 summarizes the current installations of REVS. The version at TRW is currently being optimized to reduce execution time and on-line memory requirements.

3.0 APPLICATIONS

SREM was an out-growth of research addressing the statement of software requirements for BMD systems which are fully automated (no man-machine interactions), and are real-time control and engagement oriented (engagement rules defined before software requirements definition is initiated). In addition, SREM addressed only the development of requirements for software to be executed on a single processor (no distributed processing). Although the mathematical foundations of SREM were believed to be applicable to other types of software requirements development efforts, no explicit claims were made about the efficacy of SREM for those requirements. Since the SREM approach was published, it has been applied on a variety of projects and environments. Table 4 presents characteristics of a number of these projects. Some of these projects are competitive in nature, thus sensitive details have been omitted. Results of these applications are discussed briefly below.

Projects A through E were performed at TRW under the direction of those responsible for the development of the methodology and tools. Project A involved the definition of top-level software requirements for air defense engagement control software during the conceptual design phase of the project. This project involved the definition of requirements for processing to be distributed across more than five processors, and included the definition of man/machine interactions.

Table 3 Current REVS Installations

LOCATION	HOST	I/O CAPABILITIES			
		ON-LINE GRAPHICS	REMOTE BATCH	TELETYPE	OFF-LINE GRAPHICS
ADVANCED RESEARCH CENTER, HUNTSVILLE, ALABAMA	CDC 7600	X	X		X
NAVAL RESEARCH LABS, WASHINGTON, D.C.	TI ASC			X	X
MDAC, HUNTINGTON BEACH, CALIFORNIA	CDC 7700		X		X
TRW, REDONDO BEACH, CALIFORNIA	CDC CYBER 74/174 TSS			X	X

Table 4 SREM Applications

PROJECT	PERFORMER	TYPE SYSTEM		CHARACTERISTICS			ESTIMATED SOFTWARE SIZE (INSTRUCTIONS)
				REAL-TIME	DISTRIBUTED PROCESSING	MAN/MACHINE	
A	TRW	AIR DEFENSE	CD PHASE RQMTS	X	X	X	200 K
B	TRW	OPERATING SYSTEM	RQMTS RE-DEFINITION				20 K
C	TRW	REAL-TIME INFO	RQMTS VALIDATION & RE-DEFINITION			X	200 K
D	TRW	AIR DEFENSE	DEMO	X	X		16 K
E	TRW	COMMAND/CONTROL	DEMO/RQMTS RE-DEFINITION	X	X	X	--
F	TRW	REAL-TIME INFO	CD PHASE RQMTS	X	X	X	150 K
G	TRW	RADAR CONTROL	CD DEFINE RQMTS	X			100 K
H	RCA	TEST SOFTWARE	CD DEFINE RQMTS				100 K
I	APL	COMMUNICATIONS	DEMO/RQMTS VALIDATION	X		X	50 K
J	MDAC	BMD	DEMO/DEFINE RQMTS	X			200 K
K	UCB	REACTOR CONTROL	DEMO/DEFINE RQMTS	X			--
L	TRW	RELATIONAL DB	DEMO				N/A
M	TI	RELATIONAL DB	DEMO				N/A

Project B involved the definition of the requirements for an existing operating system for which software requirements were undocumented; some extensions to the operating system were contemplated, and, hence, requirements on the existing operating system capabilities had to be definitized in order to address requirements for the augmented system.

Project C involved the application of SREM to the redefinition of a software specification for an interactive (man/machine) near real-time information management system. A major conclusion from this project was that the redefinition of software requirements follows essentially the same steps as the generation of software requirements using SREM. The major problem in performing the validation was the identification of the required sequences of processing steps defined in the various processing "functions", and identification of the data flow between processing "functions" -- just the problems that would have to be addressed to make the requirements testable.

Project D involved the redefinition and validation of requirements for existing air defense engagement software, in order to address the inclusion of requirements for new capabilities involving distributed processing. Again, the requirements were extracted from a combination of existing requirements and design documents, and sometimes from interpretation of the code itself. Conclusions from this project included the following:

- The SRE Methodology is, with little modification, applicable to requirements redefinition. The technique of first redefining the requirements, and then modifying the requirements to assess the impact on the existing software, yields significant advantages in comparison to "patching the code one more time".

- The simulation generation capability of REVS provides a rapid, cost-effective means for the generation of both functional and analytical simulations of required processing. In particular, the consistency and completeness checking capabilities of REVS speed up the process of debugging the simulation of the processing, and produce a simulation directly traceable to the requirements.

Project E involved the redefinition of top-level man/machine processing requirements for existing distributed software in order to provide an orderly means of defining augmentations, modification, and up-grade in performance requirements. This project is still on-going.

Projects F and G involved the generation of requirements for software during the conceptual development phase at TRW. Project F addressed the definition of requirements for a real-time distributed information management system, while Project G addressed requirements for software to control a radar and process its data. Both of these projects started with a customer-furnished system concept, and were to generate preliminary software requirements, a preliminary design of the software, and a fixed-price bid on the software development in about six months. Similar conclusions were arrived at in both projects:

- It is difficult to develop complete and consistent, testable software requirements on that kind of schedule with any technique, even SREM. The software requirements development process is squeezed between the system design (selecting hardware, system operating rules) and the preliminary software design (to a sufficient detail to enable a fixed-price bid). Traditionally, the specification problems are swept under a rug by writing a fuzzy software specification; SREM makes the requirements and the quality quite visible (e.g., it is entirely obvious from the automated checks of REVS whether or not all messages have been generated by the proposed functional processing, and whether or not the input/output data relationships are consistent).

- The use of SREM, particularly in the initial stages of drawing the first R-Nets, provides the communication mechanism for more meaningful discussions for clarifying the initial customer requirements. The necessity of identifying stimulus/response relationships in the R-Nets makes the top-level system logic meaningful to the user, identifies ambiguities in the initial requirements, and provides the benefits of structured walk-throughs in the very early requirements phase so the customer can identify misconceptions on the part of the requirements engineers. The customer expected perhaps a dozen questions of clarification on the requirements; to define R-Nets, he received many more questions and then commented on the R-Nets to identify misconceptions.

Project H was performed by RCA to successfully demonstrate the utility of SREM in defining requirements for test software. SREM had the effect of making the requirements more clear and understandable than the traditional methods of writing requirements for such software.

Projects I and J have just begun; no significant conclusions have yet been reached. They are, however, being carried out as demonstrations to provide information about the utility of SREM in operational environments.

In Project K, two independent teams of analysts (one academic, one industrial) wrote requirements in RSL for a portion of reactor control software, with little consultation with TRW. A significant result of the study was that the two sets of paths were identical (as predicted by the theoretical foundations).

Projects L and M took advantage of the user-extensibility features of REVS to define a new language for creating a relational data base. One significant feature of REVS is that RSL is user-extensible (i.e., new elements, attributes, and relationships can be defined by the user), and in the same (or subsequent) runs, information of this type can be translated by REVS into the automated data base, and then REVS can use these definitions to document and analyze these relationships.

3.2 EVALUATION

Only one of the above applications involved the definition of requirements for U.S. Army Ballistic Missile Defense software, the original target for SREM creation. The other projects involve SREM usage in roles not originally addressed (e.g., specification validation, specification redefinition, and augmentation), and having characteristics not originally addressed (e.g., distributed processing, man/machine interactions).

The specification validation role is a natural application for SREM, requiring very little modification of the steps, and no modification of RSL or REVS. It can be applied to a project with a preliminary software specification in order to methodically identify specification problem areas, or to a project already in software development in order to verify the completeness, consistency, and testability of the requirements. This role has been found to be a near-ideal type of project to demonstrate the utility of the SREM concepts in specific operational environments, to provide the vehicle for on-the-job training, and to provide useful results. It is one of the best mechanisms available to satisfy the new regulations (e.g., DoD Regulation 5000.29) requiring validation of software requirements before proceeding with engineering development of a large system.

The specification redefinition role is similar to the above role in that the requirements for a piece of software are defined first and then augmentations are defined with respect to the baseline specification. This approach provides significant advantages to this software augmentation process, i.e., it provides for a precise definition of the required augmentations, allows augmentations to be discussed in terms of requirements instead of specific design approaches, and provides testable requirements for the end prcduct -- the software.

Both SREM and REVS are applicable to the definition of distributed processing and man/machine interaction software requirements as they currently exist; improvements have, however, been identified. The application of SREM to these types of software provides a top-down viewpoint and testable requirements on the processing as a whole, in addition to detailed requirements directly traceable to these top-level requirements. It thus provides the framework for making the decisions for allocating the processing to the distributed nodes, for identifying communication requirements, and for allocating the total processing requirements between the software and the manual procedures. Experience with these problems has led to the identification of extensions and augmentations to RSL, REVS, and the SREM procedures. These will improve the ability to develop the detailed specifications from the top-level specifications which are directly traceable, and to provide automated support for the partitioning process.

4.0 PLANS FOR EXTENSIONS AND IMPROVEMENTS

The applications discussed in Section 3 provided experience in addressing a wider class of problems (i.e., distributed processing man/machine interactions, operating systems, on-line information systems) in various roles (i.e., specification generation, specification validation, specification redefinition and augmentation) and in various operational environments (e.g., conceptual development phase working with system and software designers vs. independent validation). Assessment of this experience has led to the identification of several types of extensions and improvements to SREM, RSL, and REVS. Specific areas of research and development currently underway include the following:

- REVS operational improvements.
- Distributed Processing Augmentations.
- Smoother transition to Process Design.
- Smoother transition from System Engineering.
- Smoother transition to Software Validation and Test Planning.
- Extensions to Business Data Processing problems.

5.0 CONCLUSIONS

There has not yet been time, since the availability of REVS, for a project to go from a conceptual design to a software specification written using the SREM technology, to a complete software design development, and test. However, from the current experiences of using SREM, in both demonstration and actual software requirements development activities, we can draw the following conclusions:

1) With the increasing availability of REVS on CDC computers, REVS is maturing to the level of capability necessary to support the definition of software requirements in operational environments.

2) SREM and REVS now have demonstrated utility in operational environments in defining and validating requirements for a wide class of software, with the domain of demonstrated applicability increasing. Augmentations to REVS have been identified to improve the capabilities of SREM to deal with an expanded class of problems.

3) The SREM technology has been successfully transferred to others for specification generation, and specification validation activities.

Thus, it appears that SREM, even after two years experience, successfully addresses the problems of defining and validating software requirements.

6.0 REFERENCES

1. M. W. Alford, "A Requirements Engineering Methodology for Real-Time Processing Requirements", IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, Jan. 1977, pp. 60-69.

2. T. E. Bell, D. C. Bixler, and M. E. Dyer, "An Extendable Approach to Computer-Aided Software Requirements Engineering", IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, Jan. 1977, pp. 49-60.

3. C. G. Davis and C. R. Vick, "The Software Development System", IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, Jan. 1977, pp. 69-84.

4. M. W. Alford and I. F. Burns, "An Approach to Stating Real-Time Processing Requirements", presented at Conf. on Petri Nets and Related Methods, Massachusetts Institute of Technology, Cambridge, MA, July 1-3, 1975.

5. V. C. Cerf, "Multi-Processors, Semaphores, and a Graph Model of Computation", Dept. of Computer Science, University of California, Los Angeles, CA, Report UCLA-ENG-7223, April 1972.

6. W. P. Stevens, G. F. Myers, and L. C. Constantine, "Structured Design", IBM Systems Journal, Vol. 13, No. 2, 1974, pp. 115-139.

7. D. Teichroew, E. Hershey, and M. Bastarache, "An Introduction to PSL/PSA", ISDOS Working Paper 86, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI, March, 1974.

8. Department of Defense, "Military Standard Specification Practices", Report MIL-STD-490, October 1968.

AUTOMATED ANALYSIS OF SYSTEM SPECIFICATIONS

Dr. Larry A. Johnson

Dr. Paul B. Merrithew

Mr. Daniel G. Smith

By LOGICON, INC. 18 Hartwell Avenue
Lexington, Massachusetts 02173

ABSTRACT

CADSAT is one of several automated tools which have been developed to help define quality requirements. This paper discusses the application of CADSAT to the analysis of high level, functional-performance specifications. A practical application of CADSAT has prompted several modifications and extensions. The CADSAT changes aid the construction of a requirements data base, help maintain the connection between the data base and other system documentation and perform special tasks. Experience reveals several essential aspects of a successful methodology for the use of the tool. This methodology and the changes to CADSAT are discussed in terms of the intrinsic capabilities of automated tools and the restrictions imposed by their use.

INTRODUCTION

The complexity of military systems development has continued to outpace the management and technical resources supporting the acquisition process. The decline in DoD manpower resources over recent years has had a two-fold effect on military systems development. The higher reliance on automated systems has been constantly increasing over the past decades and has been heightened in recent years by the desire to reduce the system's operational crew requirements and associated support functions. In conjunction with the decreasing operational personnel, the military system implementing agencies have also been impacted by staff reductions. The net effect is that systems are rising in complexity at the same time that military engineering resources responsible for the acquisition have been effectively reduced to managers of the engineering efforts. The lack of engineering staff resources is even further complicated by the lack of well-defined system engineering approaches for managing and accomplishing quality systems requirements definition and analysis (Figure 1).

Partly in response to these conditions, the Air Force Systems Command has initiated numerous programs to improve the Air Force system throughout the life cycle of the system through improved technologies and guidance. One program has been to develop automated means for defining and analyzing system requirements and encouraging the application of these computer aids in the early phases of systems acquisition. This paper describes the application of an Air Force computer aided analysis tool, CADSAT, which is being employed by Logicon system analysts in defining and analyzing the system requirements for a large surveillance system.[1] The presentation describes the goal of requirements definition and analysis, the characteristics of quality requirements, the requirements engineering process, and how automated aids are effective in accomplishing the system engineering activities for defining and analyzing system requirements.

[1] The Computer-Aided Design and Specification Analysis Tool (CADSAT) is an Air Force owned requirements analysis tool developed by the University of Michigan. The extended version is a modification developed by Logicon for applications to military systems. CADSAT's Problem Statement Language and Problem Statement Analyzer (PSL/PSA) is basically equivalent to the User Requirements Language/User Requirements Analyzer (URL/URA) developed at the University of Michigan.

COMPLEXITY OF MILITARY SYSTEMS DEVELOPMENT

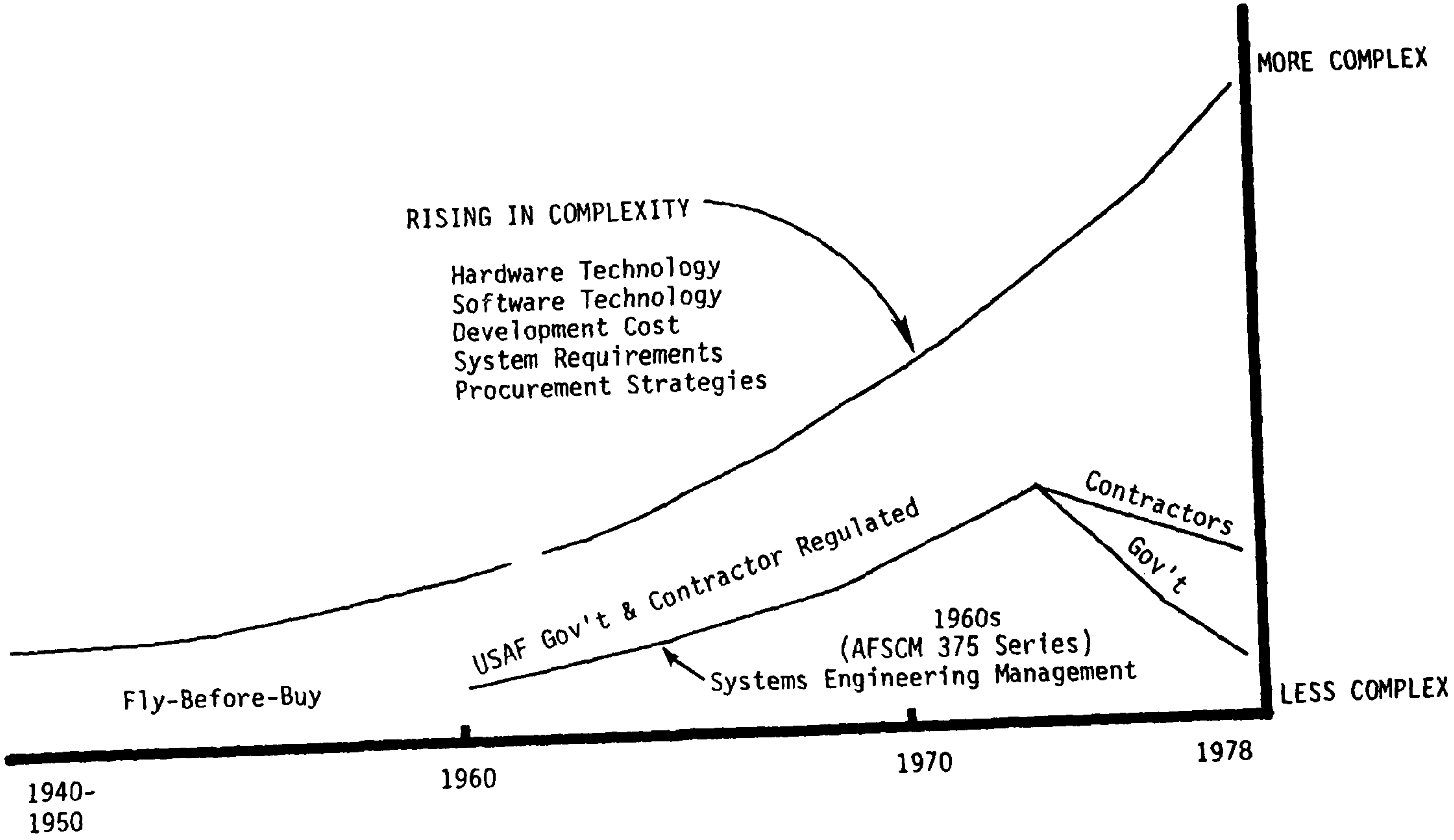


Figure 1. The complexity of military systems development has continued to outpace the management and technical resources supporting the acquisition process.

REQUIREMENTS ENGINEERING GOAL

A system in the context of this presentation is an aggregate of equipment, personnel resources, capabilities and techniques which collectively perform an operational role. The composite system includes all related facilities, items, materials, services, and personnel required for the system's self-sufficient operational deployment. System requirements engineering is concentrated upon defining the system as a whole in operational mission terms including associated performance requirements. In early system requirements engineering the analyst must avoid orientations toward specific solutions by concentrating upon defining the system in terms of what must be accomplished. As previously discussed, the lack of specific approaches and techniques for military requirements engineering allows even the best-intended analyst to digress rapidly from the "need" category to the "how-to" or solution-oriented requirements definitions. This is a natural tendency especially for any implementation-oriented engineer, such as a software engineer. Preconceived ideas from past engineering experience or operational experience with existing systems naturally come to mind. The results are "system requirements" which are semantically riddled with solution overtones or specific design details without conscious realization or justification. The thought process simply shifts to a solution oriented position (images) at the point of conceptual thought. Therefore, the requirements engineering process must recognize this tendency and must allow for effective feedbacks into the process. During the course of requirements engineering, the analyst must also be aware that non-design-oriented system documentation, such as functional-performance and development specifications, is the media for communicating the system requirements to the implementing engineers. The requirements engineering goal is to identify "discrete" requirements of the system and to "organize" these requirements in effective ways for further analysis.

QUALITY REQUIREMENTS CHARACTERISTICS

Therefore, quality requirements are dependent upon the analyst first identi-identify the discrete requirements of the system and then organizing these requirements in effective ways for further analysis (Figure 2). Initial documentation for identifying user system requirements may include early

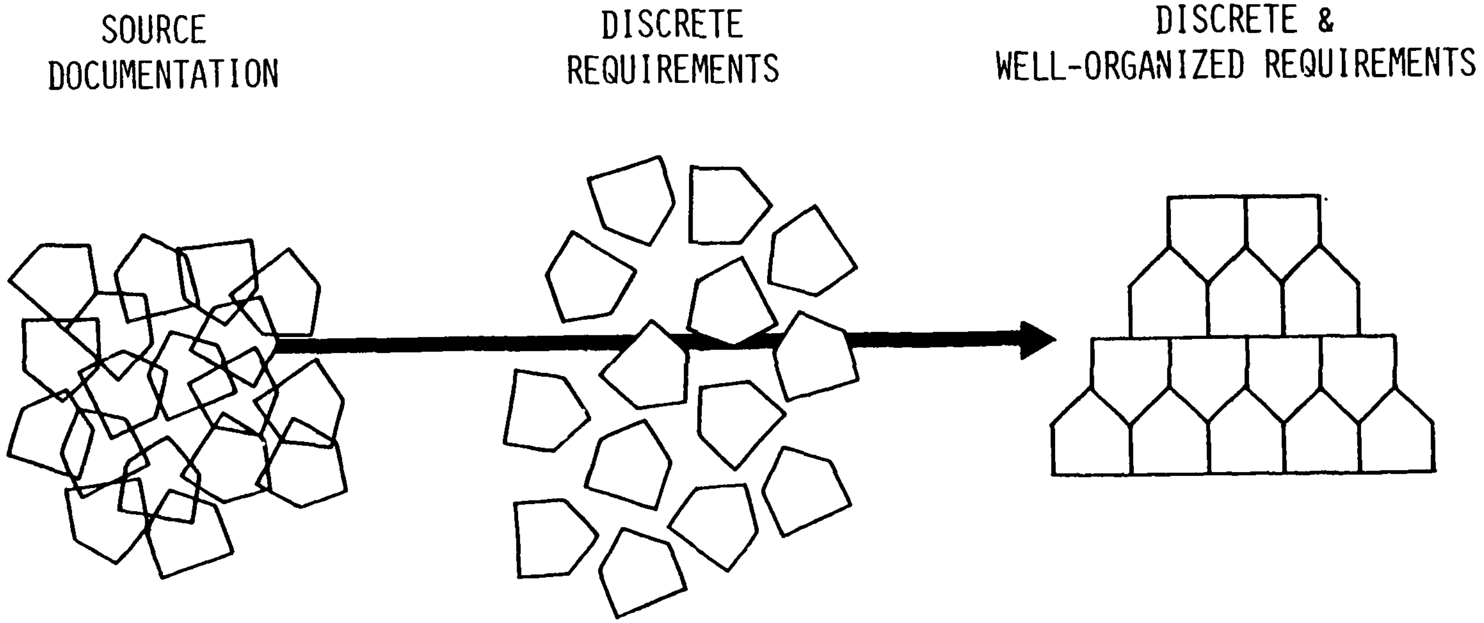


Figure 2. Develnpment of Discrete and Well-Organized Requirements

planning documents and specifications for similar systems, for system interfaces, and for existing or previously defined subsystems. In addition, documentation derived from engineering studies and prototyping or experimental test systems may be available. If the engineering activities have advanced beyond the planning and study stage, certain specifications may have already been developed. These early requirements documents have one prevailing characteristic: The system requirements are not typically distinguished (discrete) or collectively defined (well-organized). This is partly because of the fragmented nature of the early planning and study efforts which are formulative and investigatory. Another factor has been the lack of guidance in requirements engineering and the orientation of engineers to the specification documents. The specification documents in many instances become products to meet acquisition requirements and schedules as opposed to the specification being a repository of quality system requirements.

Figure 2 illustrates the first characteristic of quality requirements, the development of discrete requirements. Source documentation is analyzed and broken down in an iterative synthesis-analysis fashion throughout the requirements engineering activities. The key to identifying discrete requirements is to break the source documentation into individual parts which represent non-overlapping requirements. Requirements are categorized as functions the system must accomplish or system constraints (performance, physical, system operational effectiveness). At this point missing or incomplete requirements will begin to be more readily identified. This itemization and categorization of requirements introduces clarity where the source documentation typically is overstated, ambiguous, redundant, incomplete and inconsistent. This itemization also provides the basis for verifying the quality of the requirements and to assess the ability to test the requirements in the target system.

The second characteristic of quality requirements is the organization of the requirements in effective ways for additional analysis and for communicating them to the using agency and to implementing engineers. The identification of discrete requirements provides some awareness of omissions and gaps in the requirements. This awareness is further heightened by organizing the requirements in various ways which show the relationships between the discrete requirements (Figure 2). These relationships are logical organizational relationships, system flow relationships, and traceability relationships.

Logical organizational relationships are accomplished by structuring the discrete functions and the information requirements (external and internal input/ output) of the system into hierarchical structures. The concept of a functional hierarchical structure was introduced into military systems development through initial systems engineering practices dating back to the late 1940s. The concept has been maintained in military systems development and documentation throughout the 1960s and is an integral part of the current military standards for system documentation, i.e., MIL-STD-490 [1], MIL-STD-483 (USAF) [2], DoD 7935.1-S [3], and others. Current techniques for system development, such as the HIPO [4] visual table of contents and automated requirements analysis tools (PSL/PSA, CADSAT), retain the principles of functional hierarchical structures for communicating the functions to be accomplished by the system and the relationships between the functions. This form of organization provides a view of the system as an aggregate of functions broken out into a logical arrangement of subordinate discrete activities which must be performed. A sample portion from the Logicon-Extended Structure Report (Figure 3) demonstrates the functional break out of a surveillance system. This section of the report shows the hierarchical structure of a function (CADSAT process) at the seventh level of the hierarchy (srch-iff-data-accounting, count 1) broken out into four level 8 subordinate functions (counts 2, 3, 6 and 7). Over the course of requirements engineering many missing or incomplete functions can be directly identified from the functional hierarchical structure. Similar logical organizational relationships can be realized by structuring the internal and external information requirements of the system. The discrete system inputs, outputs (external I/O) and the internal information requirements necessary for the system's operation can be logically structured in the same manner as the functional hierarchy. The emphasis on the information structures is to arrange the information requirements into structures by breaking the information into logical subordinate parts or simply as groupings of information. The well-organized structure is effective in communicating the information requirements and for identifying incomplete or missing information requirements.

System flow relationships represent another way to organize the discrete requirements in terms of control flow and information flow. As the functions of the system are identified, the control relationships between them are

```
LOGICON EXTENDED CADSAT version  3.2r1                             04/17/78     1514.7       page
                                    Air Force ESD / RADC Multics

                                         process structure

count (level or relationship)  names      reference


    1  7 srch-iff-data-accounting
         memo segregate-simulated-data          r-3-7-1-2-3-e
    2     8 count-srch-iff                    r-3-7-1-2-3-1-1-j
            trgs site-low-data-alerts               r-3-7-1-2-3-1-1-j
            trgs site-high-data-alerts
            trgs sys-high-cond-chk-srch-iff         r-3-7-1-2-3-1-1-k
            trgs sys-overload-cond-chk-srch-iff   r-3-7-1-2-3-1-1-l
            uses radar-data-msg-in
            uses raw-search-data
            uses raw-beacon-data
            drvs raw-data-counts
            trgd limit-system-srch-iff-data         r-3-7-1-2-3-1-1-h
                                                  r-3-7-1-2-3-1-1-i
                                                  r-3-2-1-2-b
    3     8 provide-srch-iff-alerts           r-3-7-1-2-3-1-1-j
    4        9 site-low-data-alerts              r-3-7-1-2-3-1-1-j
               uses raw-data-counts
               trgd count-srch-iff                     r-3-7-1-2-3-1-1-j
    5        9 site-high-data-alerts
               uses raw-data-counts
               trgd count-srch-iff                     r-3-7-1-2-3-1-1-j
    6     8 sys-high-cond-chk-srch-iff        r-3-7-1-2-3-1-1-k
            uses raw-data-counts
            trgd count-srch-iff                     r-3-7-1-2-3-1-1-j
    7     8 sys-overload-cond-chk-srch-iff    r-3-7-1-2-3-1-1-l
            uses raw-data-counts
            trgd count-srch-iff                     r-3-7-1-2-3-1-1-j
```

Figure 3. Structure Report

identified in order to describe the logical order in which the system activities should be accomplished to satisfy the operational requirements. Figure 4 illustrates a typical CADSAT control-flow report for a portion of the surveillance system. In this report (CADSAT Process Chain) the flow of control is from left to right. Conditions which determine the flow direction when two or more branches occur are not represented in this form of CADSAT report. Control-flow analysis provides a means of viewing the system from an activity-oriented perspective and is often referred to as functional-flow analysis. As a result the requirements are viewed in a well-organized manner and missing or incomplete functions and relationships between the functions are identified. Final control-flow documentation becomes another effective means for communicating system requirements to implementing engineers. The information-flow builds upon the information hierarcy structure by providing a means of viewing the system from an information systems perspective. During this analysis the flow relationships between external system inputs and resulting outputs are identified. Quite often the most effective means of information-flow analysis is to trace an output back to system inputs, either external data, messages, or stimuli. As a result the various relationships between the associated functions and the identification of internal information necessary to support the derivation of the output are identified. Control-flow and information-flow analysis will necessitate changes and additions to previously defined functions and constraints as well as the hierarchy structures and other previously defined relationships. Missing or incomplete requirements are again determined and the deficiencies corrected.

Requirements engineering for systems which are primarily activity oriented, such as command and control systems, will naturally be concentrated on control-flow analysis as opposed to information-flow analysis. On the other hand, the system may be primarily information-processing-oriented, as a communications system or management information system. Under these circumstances the requirements engineering activities will quite naturally concentrate on information-flow analysis rather than control-flow analysis.

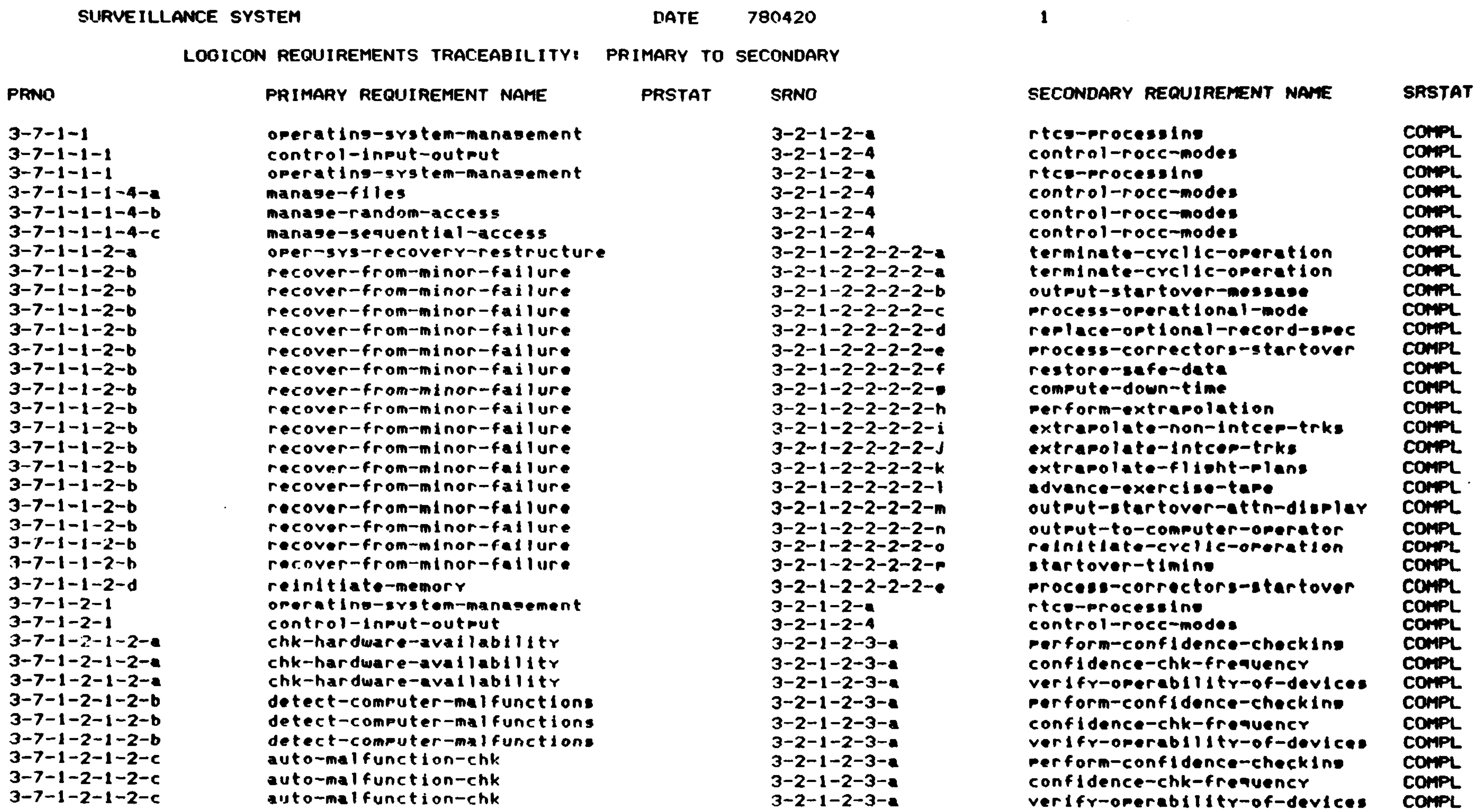
System traceability relationships are another effective means of identifying incomplete or missing requirements. During the requirements engineering activities source documents are referenced for each requirement identified.

```
LOGICON EXTENDED CADSAT version  3.2r1                           04/20/78     1057.6       page       1
                                           Air Force ESD / RADC Multics

                                                  process chain

                                                                                +--process--+       +--process--+
                                                                                |find-trans-|       |deficient- |
                                                                                |mission-er-| ..... |data-      |
                                                                               .|rors       |       |alert      |
                                                                              . +-triggered-+       +-triggered-+
                                                                             .
                                                            +--process--+  .    +--process--+
                                                            |receive-   | .     |update-    |
                                                           .|reply-     | ..... |hf-        |
                                                          . |to-dummy   | .     |status-mode|
                                                            +-triggered-+  .    +-triggered-+
                                                         .                   .
                                                                              . +--process--+
                                                        .                      .|generate-  |
                                                                                |tabular-   |
+--process--+       +--process--+       +--process--+ .                         |display    |
|filter-    |       |accept-    |       |discrimina-|                           +-triggered-+
|radar-     | ..... |height-    | ..... |te-message-|
|inputs     |       |message    |       |s          |                           +--process--+       +--process--+
+-----------+       +-triggered-+       +-triggered-+ .                         |do-        |       |control-re-|
                                                                                |error-     | ..... |al-time-in-|
                                                        .                      .|printout   |       |put-output |
                                                                              . +-triggered-+       +-triggered-+
                                                         .                   .
                                                            +--process--+  .    +--process--+       +--process--+
                                                          . |receive-   | .     |update-    |       |check-     |
                                                           .|error-     | ..... |hf-        | ..... |for-       |
                                                            |reply      | .     |operability|       |display    |
                                                            +-triggered-+  .    +-triggered-+       +-triggered-+
                                                                             .
                                                                              . +--process--+
                                                                               .|generate-  |
                                                                                |tabular-   |
                                                                                |display    |
                                                                                +-triggered-+
```

Figure 4. Control-Flow (CADSAT Process Chain), the flow is from left to right.

Traceability analysis gives the analyst a means of verifying the requirements by linking each requirement to the many and varying forms of source documentation. The Requirements Tracability Report (Figure 5) shows the traceability between specifications, as contained in separate requirements data bases. Figure 5 traces the requirements from the functional-performance specification of the surveillance system to the allocated requirements contained in the next level of specification, a development specification. This form of analysis is pertinent to validating the requirements. Relationships can also be defined to other pertinent studies, analyses, and plans which are being accomplished concurrently with the requirements engineering activities, such as program management directives and plans, system sizing and timing studies, prototyping, simulations, test planning, and the like. System test requirements (quality assurance), as well as subsequent test plans, procedures, and reports, can be effectively related to the system functional-performance requirements. The links to associated system plans, analysis, and studies accomplished prior to, during, and subsequent to the start of formal requirements engineering are crucial to the overall systems engineering concept. The traceability relationships also provide a bridge between requirements engineering activities and subsequent implementing engineering, since the requirements can be traced from functional-performance specifications to development specifications to product specificati;ns and system test activities during the later phases of the system acquisition.

Throughout the requirements engineering activities the need exists for the analyst to be able to evaluate the impact of changes to the requirements. Whatever the reason (policy, economics, study or analysis results, engineering change proposals, etc.), the analyst must be in a position to determine the ramifications to changes in the system requirements. Once the area of impact is identified in the system requirements, (functions, constraints, control-flow and data-flow relationships, etc.) the traceability relationships provide the capability to readily identify associated impacts to the system as well as trace the impacts to all other associated documentation such as the program directives, plans, studies and analyses, test plans, associated system specifications (functional-performance, development, product) and the like. The impact can be determined with exacting visibility, and the appropriate actions taken.



```
SURVEILLANCE SYSTEM                                  DATE    780420                    1

                LOGICON REQUIREMENTS TRACEABILITY:  PRIMARY TO SECONDARY

PRNO                PRIMARY REQUIREMENT NAME        PRSTAT     SRNO               SECONDARY REQUIREMENT NAME     SRSTAT

3-7-1-1             operating-system-management                3-2-1-2-a          rtcg-processing                COMPL
3-7-1-1-1           control-input-output                       3-2-1-2-4          control-rocc-modes             COMPL
3-7-1-1-1           operating-system-management                3-2-1-2-a          rtcg-processing                COMPL
3-7-1-1-1-4-a       manage-files                               3-2-1-2-4          control-rocc-modes             COMPL
3-7-1-1-1-4-b       manage-random-access                       3-2-1-2-4          control-rocc-modes             COMPL
3-7-1-1-1-4-c       manage-sequential-access                   3-2-1-2-4          control-rocc-modes             COMPL
3-7-1-1-2-a         oper-sys-recovery-restructure              3-2-1-2-2-2-2-a    terminate-cyclic-operation     COMPL
3-7-1-1-2-b         recover-from-minor-failure                 3-2-1-2-2-2-2-a    terminate-cyclic-operation     COMPL
3-7-1-1-2-b         recover-from-minor-failure                 3-2-1-2-2-2-2-b    output-startover-message       COMPL
3-7-1-1-2-b         recover-from-minor-failure                 3-2-1-2-2-2-2-c    process-operational-mode       COMPL
3-7-1-1-2-b         recover-from-minor-failure                 3-2-1-2-2-2-2-d    replace-optional-record-spec   COMPL
3-7-1-1-2-b         recover-from-minor-failure                 3-2-1-2-2-2-2-e    process-correctors-startover   COMPL
3-7-1-1-2-b         recover-from-minor-failure                 3-2-1-2-2-2-2-f    restore-safe-data              COMPL
3-7-1-1-2-b         recover-from-minor-failure                 3-2-1-2-2-2-2-g    compute-down-time              COMPL
3-7-1-1-2-b         recover-from-minor-failure                 3-2-1-2-2-2-2-h    perform-extrapolation          COMPL
3-7-1-1-2-b         recover-from-minor-failure                 3-2-1-2-2-2-2-i    extrapolate-non-intcep-trks    COMPL
3-7-1-1-2-b         recover-from-minor-failure                 3-2-1-2-2-2-2-J    extrapolate-intcep-trks        COMPL
3-7-1-1-2-b         recover-from-minor-failure                 3-2-1-2-2-2-2-k    extrapolate-flight-plans       COMPL
3-7-1-1-2-b         recover-from-minor-failure                 3-2-1-2-2-2-2-l    advance-exercise-tape          COMPL
3-7-1-1-2-b         recover-from-minor-failure                 3-2-1-2-2-2-2-m    output-startover-attn-display  COMPL
3-7-1-1-2-b         recover-from-minor-failure                 3-2-1-2-2-2-2-n    output-to-computer-operator    COMPL
3-7-1-1-2-b         recover-from-minor-failure                 3-2-1-2-2-2-2-o    reinitiate-cyclic-operation    COMPL
3-7-1-1-2-b         recover-from-minor-failure                 3-2-1-2-2-2-2-p    startover-timing               COMPL
3-7-1-1-2-d         reinitiate-memory                          3-2-1-2-2-2-2-e    process-correctors-startover   COMPL
3-7-1-2-1           operating-system-management                3-2-1-2-a          rtcg-processing                COMPL
3-7-1-2-1           control-input-output                       3-2-1-2-4          control-rocc-modes             COMPL
3-7-1-2-1-2-a       chk-hardware-availability                  3-2-1-2-3-a        perform-confidence-checking    COMPL
3-7-1-2-1-2-a       chk-hardware-availability                  3-2-1-2-3-a        confidence-chk-frequency       COMPL
3-7-1-2-1-2-a       chk-hardware-availability                  3-2-1-2-3-a        verify-operability-of-devices  COMPL
3-7-1-2-1-2-b       detect-computer-malfunctions               3-2-1-2-3-a        perform-confidence-checking    COMPL
3-7-1-2-1-2-b       detect-computer-malfunctions               3-2-1-2-3-a        confidence-chk-frequency       COMPL
3-7-1-2-1-2-b       detect-computer-malfunctions               3-2-1-2-3-a        verify-operability-of-devices  COMPL
3-7-1-2-1-2-c       auto-malfunction-chk                       3-2-1-2-3-a        perform-confidence-checking    COMPL
3-7-1-2-1-2-c       auto-malfunction-chk                       3-2-1-2-3-a        confidence-chk-frequency       COMPL
3-7-1-2-1-2-c       auto-malfunction-chk                       3-2-1-2-3-a        verify-operability-of-devices  COMPL
```

Figure 5. Requirements Traceability Report

Discrete and well-organized requirements support the primary goal of defining the operational mission needs of the using activity while giving the analyst visibility and control over the system definition process. Discrete and well-organized requirements become effective prerequisites for the creation of system functional-performance or development specifications. These documents then achieve a secondary requirements engineering goal of communicating the system requirements as a whole to implementing engineers.

REQUIREMENT ENGINEERING

The quality of system requirements balances upon a requirements engineering methodology as illustrated in Figure 6. The requirements engineering methodology is a well-defined approach for defining and analyzing the system's functional-performance requirements. The methodology incorporates the requirements engineering procedural issues, tools and techniques, and products, usually in the form of specification documentation. Each application of the methodology necessitates considerations for management visibility and control, qualified systems analysis, analyst team organization, and specific applications planning. The requirements engineering methodology applied by Logicon to military systems has principally evolved from work on a surveillance system currently under development. The application of automated aids in the surveillance system development has resulted in Logicon extending and modifying CADSAT for initial systems definition and analysis activities in order to provide the analyst with requirements analysis capabilities that were otherwise unavailable. The approach taken in identifying and analyzing the surveillance system has been refined and applied to other Logicon contracts.

Two factors which influenced the application of automated aids to the surveillance project were previously reported by Johnson and Merrithew [5]. First, the requirements were analyzed at very high levels and the computer maintained requirements data contained only the high level requirements of the surveillance system. Second, the large number of requirements for the surveillance system had to be analyzed completely with limited personnel and computer resources. As a result the basic approach was to keep the methodology as direct and uncomplicated as possible. The requirements data base served as a repository of high level requirements which were organized to aid the analyst

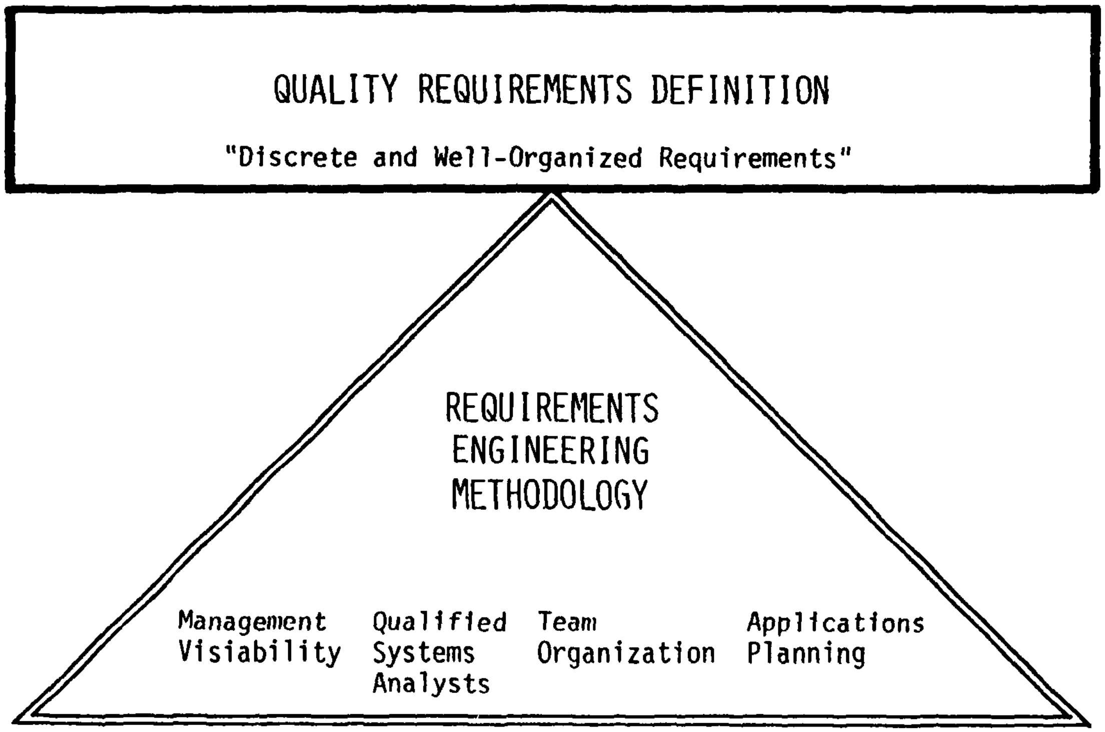
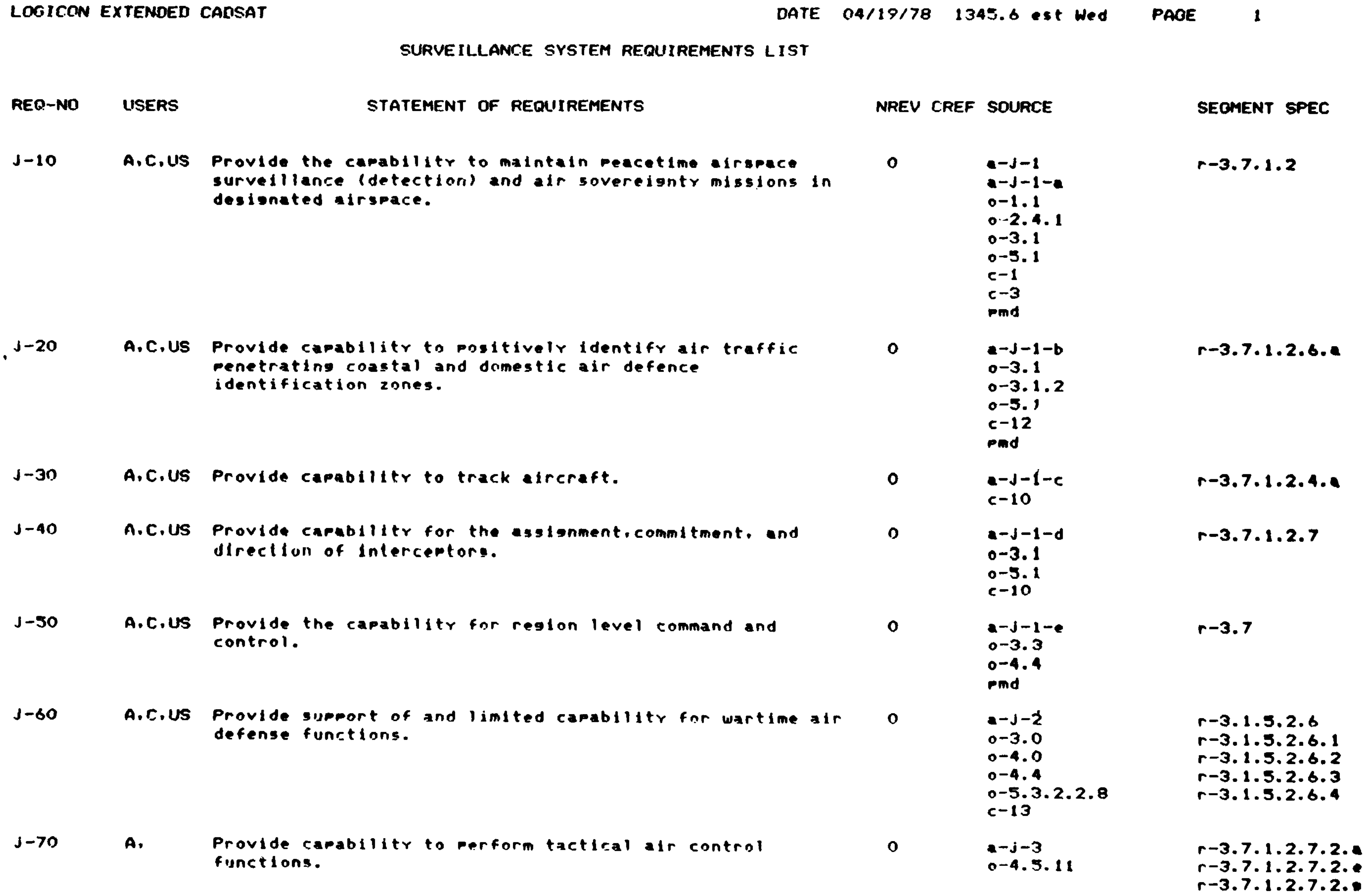

QUALITY REQUIREMENTS DEFINITION
"Discrete and Well-Organized Requirements"
REQUIREMENTS ENGINEERING METHODOLOGY
Management Visiability
Qualified Systems Analysts
Team Organization
Applications Planning

Figure 6. The Quality of System Requirements Balances Upon a Requirements Engineering Methodology.

in answering key questions about the system. As such the data base repository was not necessarily developed to provide detailed answers to every system requirement. In addition, some liberty was taken with several CADSAT features in order to achieve a quality requirements system definition. The extensions and modifications of CADSAT were primarily oriented toward increasing the analyst's visibility into the evolving requirements data base.

Two source documents were analyzed for the surveillance system. One requirement document was an early user requirements list (Figure 7) which preceded the development of the system functional-performance specification (system specification). The principle requirements, however, were derived from a 500-page system specification. Traceability was maintained between the two source documents throughout the requirements engineering activities.

The actual requirements engineering activities for the surveillance system proceeded in a series of steps [5]. The initial step identified the discrete requirements of the system and categorized them by type (CADSAT Objects), such as functions and constraints. The requirements were given unique short descriptive names such as radar-input-capacity-timing (line 1 of Figure 8). The *naming convention was consistent throughout the development of the requirements* data base and was predicated upon giving a subject-verb meaning to the object name. In addition synonyms were established for each object to aid in further analysis and to facilitate the format requirements of certain automated reports (Figure 8). Over the course of the initial requirements engineering activities the logical organizational relationships (functional hierarchies and information hierarchies) were entered into the CADSAT requirements data base. Inconsistent and incomplete requirements were identified by the analyst as being difficult or impossible to integrate into the logical hierarchies. Redundancies in the source document were realized as similar names or synonyms were given to other CADSAT data base object names already in the hierarchy. Reports such as the structure report (Figure 3) were used to determine the completeness of the definition of the system functional requirements. As incomplete requirements were realized, the synthesis-analysis process continued and discrepencies were corrected.



```
LOGICON EXTENDED CADSAT                                              DATE  04/19/78  1345.6 est Wed     PAGE     1

                                      SURVEILLANCE SYSTEM REQUIREMENTS LIST

REQ-NO     USERS                 STATEMENT OF REQUIREMENTS                   NREV CREF SOURCE            SEGMENT SPEC

J-10       A,C,US   Provide the capability to maintain peacetime airspace       0      a-J-1             r-3.7.1.2
                    surveillance (detection) and air sovereignty missions in           a-J-1-a
                    designated airspace.                                               o-1.1
                                                                                       o-2.4.1
                                                                                       o-3.1
                                                                                       o-5.1
                                                                                       c-1
                                                                                       c-3
                                                                                       pmd

J-20       A,C,US   Provide capability to positively identify air traffic       0      a-J-1-b           r-3.7.1.2.6.a
                    penetrating coastal and domestic air defence                       o-3.1
                    identification zones.                                              o-3.1.2
                                                                                       o-5.1
                                                                                       c-12
                                                                                       pmd

J-30       A,C,US   Provide capability to track aircraft.                       0      a-J-1-c           r-3.7.1.2.4.a
                                                                                       c-10

J-40       A,C,US   Provide capability for the assignment,commitment, and       0      a-J-1-d           r-3.7.1.2.7
                    direction of interceptors.                                         o-3.1
                                                                                       o-5.1
                                                                                       c-10

J-50       A,C,US   Provide the capability for region level command and         0      a-J-1-e           r-3.7
                    control.                                                           o-3.3
                                                                                       o-4.4
                                                                                       pmd

J-60       A,C,US   Provide support of and limited capability for wartime air   0      a-J-2             r-3.1.5.2.6
                    defense functions.                                                 o-3.0             r-3.1.5.2.6.1
                                                                                       o-4.0             r-3.1.5.2.6.2
                                                                                       o-4.4             r-3.1.5.2.6.3
                                                                                       o-5.3.2.2.8       r-3.1.5.2.6.4
                                                                                       c-13

J-70       A,       Provide capability to perform tactical air control          0      a-J-3             r-3.7.1.2.7.2.a
                    functions.                                                         o-4.5.11          r-3.7.1.2.7.2.e
                                                                                                         r-3.7.1.2.7.2.g
```

Figure 7. Requirements Listing Report

```
SURVEILLANCE SYSTEM        REV-PERIOD  30       DATE  04/18/78  1425.6 est Tue       PAGE

                                  SPECIFICATION DATA BASE STATUS REPORT

REQ. NO.            REQUIREMENT NAME                  SYNONYM   TYPE   CAT   SEQN STAT   NREV  SY DS SR KW TK UD RO TT PS UU MM OI

3-7-1-2-3-a         radar-input-capacity-timing       rainca    memo   surv     4 COMPL  0     1  0  0  2  0 00 00 00 16 00  2
3-7-1-2-3-b         process-simulated-radar-data      prsira    proc   surv    31 COMPL  0     1  0  0  1  0 00 00 00 13 00  0
                    process-in-std-operations-only    prstop    proc   surv    32 COMPL  0     1  0  0  1  0 00 00 00 10 00  0
3-7-1-2-3-c         process-in-simulation-over-lie    prsili    proc   surv    33 COMPL  0     1  0  0  3  0 00 00 00 10 00  0
3-7-1-2-3-d         process-in-simulated-exercise     prsiex    proc   surv    34 COMPL  0     1  0  0  1  0 00 00 00 10 00  0
3-7-1-2-3-e         segregate-simulated-data          sesida    proc   surv    35 COMPL  0     1  0  0  1  0 00 00 00 10 00  0
3-7-1-2-3-f         radar-input-capacity-timing       rainca    memo   surv     4 COMPL  0     1  0  0  2  0 00 00 00 16 00  2
3-7-1-2-3-g         filter-radar-inputs               firame    proc   surv     5 COMPL  0     1  0  0  1  0 21 00 16 10 00  0
3-7-1-2-3-i         accept-sim-via-priority           acsipr    proc   surv    36 COMPL  0     1  0  0  2  0 00 00 00 10 00  0
3-7-1-2-3-1-1-a     accept-normal-radar-data          acnora    proc   surv     6 COMPL  0     2  0  0  1  0 00 00 00 13 00  2
3-7-1-2-3-1-1-b     set-status-srch-iff               sestsr    proc   surv    47 COMPL  0     1  0  0  1  0 11 00 00 10 00  0
                    accept-sim-via-priority           acsipr    proc   surv    36 COMPL  0     1  0  0  2  0 00 00 00 10 00  0
3-7-1-2-3-1-1-c     map-srch-iff                      masrif    proc   surv     7 COMPL  0     1  0  0  2  0 21 00 11 10 00  0
3-7-1-2-3-1-1-d     inputs-in-ptc-srch-iff            inptsr    memo   surv     6 COMPL  0     1  0  0  1  0 00 00 00 13 00  2
3-7-1-2-3-1-1-e     flag-emergency-codes              flemco    proc   surv    18 COMPL  0     1  0  0  1  0 11 00 12 10 00  0
3-7-1-2-3-1-1-f     acceptance-capacity-srch-iff      accasr    memo   surv     6 COMPL  0     1  0  0  1  0 00 00 00 13 00  2
3-7-1-2-3-1-1-g     limit-set-srch-iff-data           lissid    proc   surv     9 COMPL  0     1  0  0  1  0 21 00 11 10 00  0
3-7-1-2-3-1-1-h     limit-system-srch-iff-data        lissif    proc   surv    10 COMPL  0     1  0  0  3  0 12 00 12 10 00  0
3-7-1-2-3-1-1-i     limit-system-srch-iff-data        lissif    proc   surv    10 COMPL  0     1  0  0  3  0 12 00 12 10 00  0
3-7-1-2-3-1-1-j     count-srch-iff                    cosrif    proc   surv    12 COMPL  0     1  0  0  1  0 31 00 14 10 00  0
                    provide-srch-iff-alerts           prsrif    proc   surv    13 COMPL  0     1  0  0  1  0 00 00 00 12 00  0
                    site-low-data-alerts              siloda    proc   surv    14 COMPL  0     1  0  0  1  0 11 00 10 10 00  0
3-7-1-2-3-1-1-k     sys-high-cond-chk-srch-iff        shccsi    proc   surv    16 COMPL  0     1  0  0  1  0 11 00 10 10 00  0
3-7-1-2-3-1-1-l     sys-overload-cond-chk-srch-iff    soccsi    proc   surv    17 COMPL  0     1  0  0  1  0 11 00 10 10 00  0
3-7-1-2-3-1-2       bias-adjustment-srch-iff          biadsr    proc   surv    51 COMPL  0     1  0  0  1  0 33 00 12 10 00  0
```

Figure 8. Data Base Status Report

AD-A104 252 INTERNATIONAL BUSINESS SERVICES INC WASHINGTON DC F/G 9/2
US ARMY SOFTWARE SYMPOSIUM (2ND) HELD AT WILLIAMSBURG, VIRGINIA--ETC(U)
1978 S M TAYLOR DAAK70-78-D-0030

UNCLASSIFIED NL

4 OF 10
AD
A104252

The second level of analysis for the surveillance system identified the system flow relationships, i.e., contol flows and information flows. This analysis proceeded as previously described. The information flow revealed significant ill-defined and ambiguous requirements as a result of the analyst being unable to identify the information flow within the source documentation.

The final step for analyzing the surveillance system identified the relationships between the CADSAT-maintained requirements data base and the source documentation. Once these relationships were identified, CADSAT trace keys were added to the requirements data base. The Requirements Traceability Report (Figure 5) presents the requirements traceability relationships in a convenient format while providing the analyst with a complete list of requirements (not shown) which remain to be traced. The application of CADSAT to the surveillance system is being continued by Logicon. Analysis activities are being performed on computer program development and product specifications in order to establish the traceability of the system requirements from one requirements data base to another (i.e., functional performance to development to product specifications).

This continuing traceability analysis provides the additional capability to evaluate the impact of changes to requirements. For example, the impact of a change to a paragraph in the source documentation such as the surveillance system functional-performance specification can be evaluated through the use of a series of CADSAT-generated reports beginning with the Formatted Problem Statement Report (Figure 9). This example shows all the requirement types (CADSAT Objects) relative to the specified paragraph number (line 3, "keyword" r-3-7-1-2-3-1-1-j is a reference to the system specification paragraph 3.7.1.2 3.1.1j). Once the requirement has been identified, any of several reports can be generated which may reveal possible impacts. As any military system acquisition continues, system developers propose changes to the requirements as technical issues are identified. These engineering change proposals (ECPs) can be evaluated using similar analytical techniques and CADSAT report generation capabilities. ECP impact analysis is enhanced by the traceability relationships which link the acquisition specifications together through various requirements data bases.

```
LOGICON EXTENDED CADSAT version  3.2r1                      05/08/78     1636.1
                                         Air Force ESD / RADC Multics

                                         formatted problem statement

   1 process                                         count-srch-iff;
   2    synonyms are:    cosrif;
   3    keywords:        r-3-7-1-2-3-1-1-j;
   4    attributes are:
   5          location         1.1.1.1.2.2.3.1,
   6          schedule         g0140124006----------datpr-479;
   7    part of:         srch-iff-data-accounting;
   8    uses:            radar-data-msg-in,
   9                     raw-search-data,
  10                     raw-beacon-data;
  11    derives:         raw-data-counts;
  12    triggers:        site-low-data-alerts,
  13                     site-high-data-alerts,
  14                     sys-high-cond-chk-srch-iff,
  15                     sys-overload-cond-chk-srch-iff;
  16    triggered by:    limit-system-srch-iff-data;
```

Figure 9. Formatted Problem Statement Report

AUTOMATED AIDS

The emphasis of this presentation has been on the goal of requirements engineering, the characteristics of quality requirements, and the principles and practices of requirements engineering. Automated aids such as CADSAT must be viewed only as tools in defining and analyzing system requirements. Automated aids usually lack many of the capabilities which are necessary for early requirements definition and therefore require modifications and further development. The origins of these tools within academic and R&D envionments and the lack of pragmatic applications to complex military systems development are evident in the performance and design of the initial tools. Logicon CADSAT extensions have been effective and additional needs have been identified but have not been implemented because of resource limitations. These needs are currently being documented in a Logicon study in which the requirements for automated aids are being defined within the context of a standard for requirements engineering (Figure 10). Many of the issues addressed in this paper will be incorporated into the standard. As illustrated in Figure 10, requirements analysis tools like CADSAT can be effective in the early acquisition phases of system development in conjunction with standards for requirements engineering.

The general capabilities of requirements definition and analysis tools are to (1) provide a medium for formal definition of requirements, (2) perform rudimentary analysis, (3) allow for a flexible and iterative approach to requirements engineering, and (4) to produce documentation. The formalism is provided by the automated tool's ability to evaluate the syntax of each requirement defined and to maintain and manipulate the requirements data base. Language features and other characteristics naturally encourage rigor where manual methods are less exacting. Requirements definition and analysis features of current automated tools provide the primary benefits of quickly correlating the requirements and relationships. Additional analysis capabilities are assuring compliance with the requirements language, manipulating raw source documentation requirements, and identifying incomplete requirements.

Automated tools must allow for an evolutionary technical definition of the system. Each system has unique system engineering management problems which

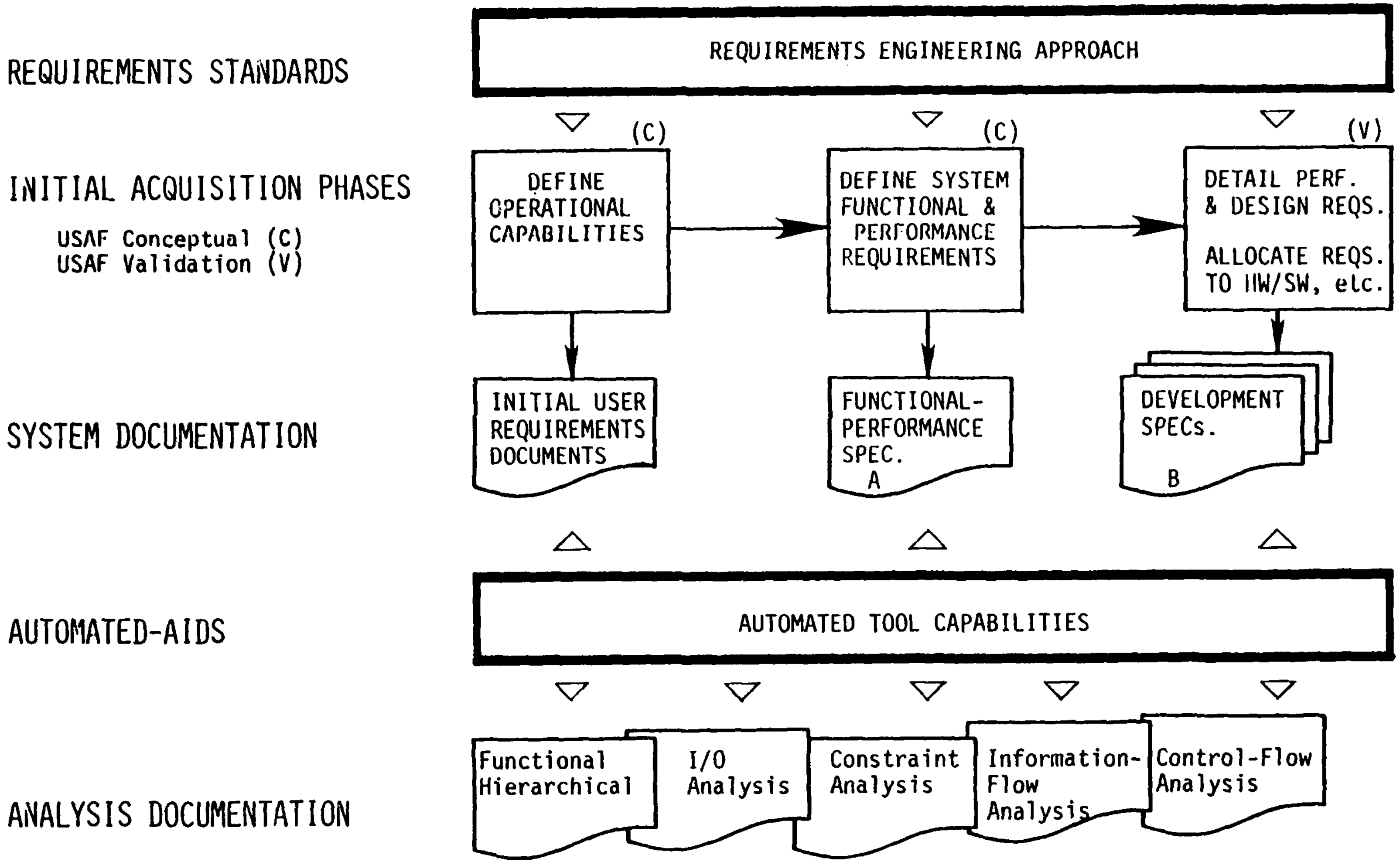

REQUIREMENTS STANDARDS
INITIAL ACQUISITION PHASES
USAF Conceptual (C)
USAF Validation (V)
SYSTEM DOCUMENTATION
AUTOMATED-AIDS
ANALYSIS DOCUMENTATION
REQUIREMENTS ENGINEERING APPROACH
(C)
DEFINE OPERATIONAL CAPABILITIES
(C)
DEFINE SYSTEM FUNCTIONAL & PERFORMANCE REQUIREMENTS
(V)
DETAIL PERF. & DESIGN REQS.
ALLOCATE REQS. TO HW/SW, etc.
INITIAL USER REQUIREMENTS DOCUMENTS
FUNCTIONAL-PERFORMANCE SPEC.
A
DEVELOPMENT SPECs.
B
AUTOMATED TOOL CAPABILITIES
Functional Hierarchical
I/O Analysis
Constraint Analysis
Information-Flow Analysis
Control-Flow Analysis

Figure 10. Automated Tool Capabilities are being defined by Logicon within the Context of a Standard for Requirements Engineering.

evolve over the course of the early acquisition phases. Because of the confusion of requirements documentation and the technical and management direction of the early acquisition phases of military systems, automated aids must permit a natural flexible iterative approach which recognizes the complexity of issues which must be addressed. Finally, the aid must provide a means for intermediate and final documentation of system requirements in timely, up-to-date and easy-to-read formats. Additional CADSAT capabilities for impact analysis (specification changes, ECPs, etc.), traceability between specifications and system testing documentation, configuration control of system documentation, and automated specification updating are currently being addressed by Logicon.

CADSAT provides benefits to the requirements engineering approach in several ways. CADSAT assures a certain rigor is achieved in the requirements engineering activities which other techniques must enforce manually. The analyst's technical perspective is improved by the ability to define the discrete requirements of the system and to organize them into effective representations. CADSAT provides a means for identifying inconsistent and incomplete requirements and representing and correlating the requirements of very large systems; this would not be practical by manual methods. CADSAT reduces many of the clerical problems which burden the analyst, such as providing up-to-date working documentation, revised specification requirements, and identifying and correcting clerical mistakes. The lack of this clerical burden allows the analyst to concentrate on the requirements definition and analysis activities. Three other benefits of CADSAT analysis have been to reduce the time for analysis in the surveillance system activities (Figure 11), increase confidence in the results of the analysis, and introduce technical management visibility into the requirements engineering activities.

	PROBLEM	CADSAT ASSISTED TIME (HOURS)	ESTIMATED TIME W/O CADSAT (HOURS)	ESTIMATED TIME SAVED (%)
1)	IDENTIFY LATERAL-TELL MESSAGE INCONSISTENCY	4	24	83
2)	IDENTIFY EXTERNAL INTERFACE REQUIREMENTS	1	16	94
3)	DETERMINE IMPACT OF 250 REQUIREMENTS CHANGES	32	120	73
4)	IDENTIFY TRACKING ALGORITHM ERROR	*	(*)(+)	--
5)	IDENTIFY IMPACT OF CHANGING OPERATOR	12	40	70
6)	ANALYZE BASELINE REQUIREMENTS	24	(*)(+)	--
7)	ANALYZE FUNCTIONAL-PERFORMANCE SPEC	160	800	80
8)	IDENTIFY EFFECTS OF CHANGES	24	(*)(+)	--

NOTES:

(*) HOURS CANNOT BE DISTINGUISHED FROM OTHER REQUIRED TASKS

(+) WOULD NOT BE IDENTIFIED DURING EARLY REQUIREMENTS ENGINEERING, I.E., USUALLY ACCOMPLISHED DURING SYSTEM INTEGRATION AND TEST

FIGURE 11. EXAMPLES OF CADSAT BENEFITS ON SURVEILLANCE SYSTEM REQUIREMENTS ENGINEERING ACTIVITIES BEING PERFORMED BY LOGICON

REFERENCES

[1] MIL-STD-490, "Specification Practices," 30 October 1968.

[2] MIL-STD-483 (USAF), "Configuration Management Practice for Systems, Equipment, Munitions, and Computer Programs," 31 December 1970.

[3] DoD 7935.1-S, "Automated Data Systems Documentation."

[4] "HIPO - A Design Aid and Documentation Technique," IBM Manual, 1974.

[5] L. A. Johnson and P. B. Merrithew, "Automated Support for Requirements Analysis and Traceability," Logicon Paper, IEEE 1978 National Aerospace and Electronics Conference, 18 May 1978.

PATRIOT SOFTWARE SYSTEM

Edward U. Lee
Rayethon Company

PATRIOT SOFTWARE DEVELOPMENT

SESSION CHAIRPERSON: Edward U. Lee, Jr.

PATRIOT Software Development
Raytheon Company Missile Systems Company

SESSION SUMMARY

PATRIOT is a highly automated air defense system designed to combat the air defense threat of the 1980's and 1990's. This presentation provided an overview of the functions, management, development, and validation of the operational software for PATRIOT.

The presentation began with an operational description of PATRIOT, its advanced multiprocessor computer system, and its operational software. Operating in a floating executive environment, the software resident in the PATRIOT weapon control computer manages and controls the system's initialization, surveillance, guidance, communication, and command and coordination functions. Operator intervention is provided via a situation display, which also presents a real time summary of system status. After discussing the software and its functions, the presentation went on to describe the PATRIOT software development cycle.

Initially, support and diagnostic software development was paralleled by an extensive systems analysis effort that led to the production of detailed software performance requirements. When the requirements were placed under configuration control, the software functional design began, and the requirements were allocated to over a hundred program units. After the individual program units were designed, coded and acceptance tested on a UNIVAC 1108-based simulation of the target multiprocessor, the recomposition process began. Groups of program units were integrated with a real time executive program into a series of software builds (software ensemble that provided significant levels of control of PATRIOT real time operations, such as surveillance, display or guidance control). The functional builds were then acceptance-tested on the PATRIOT multiprocessor computer system, operating under the stimulus of a comprehensive real time computer simulation of the PATRIOT system and its environment. This simulation provided a relatively complete dynamic test environment, with a large degree of scenario variability.

Build testing via simulation resulted in the detection and correction of approximately ninety percent of the software "bugs" prior to the start of hardware/software integration at the system level. Subsequent PATRIOT surveillance and flight text operations provided feedback to enhance the utility of the software test bed capability.

The final software builds, obtained by the sequential integration and test of the earlier builds, required over 200,000 words of on-line storage. The total PATRIOT software developed to date approximates 3,000,000 executable instructions, which were developed using a comprehensive in-process software documentation and configuration management system. This software development system is in many senses prototypical of the latest DOD software management initiatives, and contains many features applicable to both large and small scale software development efforts.

TEST BEDS

John M. Cole
CENTACS

TEST BEDS

SESSION CHAIRPERSON: John M. Cole

System Validation Division
CENTACS

SESSION SUMMARY

This session discussed how the Teleprocessing Design Center (TDC) test bed supports experimentation in the development and validation of Army tactical data systems, evaluates technical concepts and performs interoperability experiments so that technical data can be gathered for detailed analysis.

The first paper dealt with the overall establishment of the TDC where, with on-line emulation, simulation, contemporary communications equipment, and the use of actual tactical equipments, systems could be tested and evaluated in a benign environment where repeatability scenarios could be executed and dat on performance could be gathered and analyzed. The use of a family of equipment and emulation/simulation of non-available devices allows reconfiguration of current/proposed systems in an affordable manner. The TDC laboratory test equipment, library of emulators and scenarios, allows data to be taken and analyzed to provide the development system manager with the criteria to make management decisions which in turn provide the Army with cost effective, austere, automatic command and control systems.

The second paper dealt with interfacing the system within the TDC to the ARPANET. The Army has a requirement to achieve interoperability of its systems, both intra-Army and with other services and defense agency systems (JINTACCS Program). The Communication Research and Development Command (CORADCOM) teleprocessing Design Center (TDC) at Fort Monmouth has proven that emulation is a viable tool to validate the performance of computerized systems.

The Army plans to use the TDC to support Army interoperability validation. However, there may be cases where emulation of systems of interoperable sets may not be practical, desirable or even feasible. To connect or utilize actual tactical equipment for interoperability testing, a method of connection to the TDC is necessary. The Army plans to connect remotely located tactical systems to the TDC via the ARPANET to permit the interconnection of emulated and actual tactical systems.

Toward this end, the Army is developing a special general purpose interface unit that will provide hardware, software and protocol combatibility between tactical systems (real or emulated) and the ARPANET's communication computer, the Interface Message Processor (IMP). In addition, remote users will be able to develop, test and check out their software on the TDC's emulation system.

The third paper dealt with the Software Development Support System (SDSS) which is a planned Government-owned computing facility configured to satisfy the program development needs of users/contractors engaged in the MCF-TOS demonstration model development effort. SDSS is intended to provide a centrally located facility of commercial off-the-shelf hardware and software augmented by a variety of advanced, specialized software tools required to support the various life cycle development activities of militarized computer systems. SDSS will provide a working environment where such activities as configuration management, test, verification and validation, training and post-deployment support can be conducted. The SDSS will address the problems of uncontrolled tactical software development arising from non-government ownership of development software and the proliferation of development tools. SDSS will attempt to reconcile the conflicting requirements of support software developers and support software users.

Emulation of Tactical Data Systems in the Teleprocessing Design Center

John M. Cole

CENTACS

This presentation dealt with the overall establishment of the Teleprocessing Design Center (TDC) where, with on-line emulation, simulation, contemporary communications equipment, and the use of actual tactical equipment, systems could be tested and evaluated in a benign environment where repeatable scenarios could be executed and data on performance could be gathered and analyzed. The use of a familv of equipment and emulation/simulation of non-available devices allows reconfiguration of current proposed systems in an affordable manner. The TDC laboratory test equipment, library of emulators and scenarios, allows data to be taken and analyzed to provide the development system manager with the criteria to make management decisions which in turn provide the Army with cost-effective, austere, automatic command and control systems.

EMULATION OF TACTICAL DATA SYSTEMS IN THE TELEPROCESSING DESIGN CENTER

John M. Cole

US Army Communications Research
and Development Command
Center for Tactical Computer Systems
Systems Validation Division
Fort Monmouth, New Jersey

INTRODUCTION

The Teleprocessing Design Center (TDC) was established within the Center for Tactical Computer Systems (CENTACS) at Fort Monmouth, New Jersey, for the purpose of supporting experimentation in the definition, development, acquisition, and validation of individual tactical computing systems and to support the validation of interoperability among tactical data systems.

The use of high speed digital computers, digital entry message devices, remote battlefield sensors, and other state-of-the-art devices in real time command and control systems has presented the system developer with many complex problems that have lengthened the development cycle which, in turn, makes the systems expensive and dated before they are fielded.

One effort to resolve technical uncertainties and ambiguties, and to validate technical concepts prior to building hardware, was the activation of the TDC where, with on-line simulation, emulation, contemporary communications equipment and the use of actual tactical equipment, systems could be tested and evaluated in a benign environment where repeatable scenarios could be executed and data on performance could be gathered and analyzed. The use of a family of equipment and emulation/simulation of non-available devices allows reconfiguration of current/proposed systems in an affordable manner. The TDC laboratory test equipment and library of emulators and scenarios allows data to be taken and analyzed to provide the development system manager with the criteria to make management decisions which, in turn, provide the Army with cost effective, austere, automatic command and control systems.

MISSION AND FUNCTIONS OF THE TDC

The mission of the TDC is to support experimentation in the development and validation of Army tactical data systems, to test and evaluate technical concepts, systems integration, and to perform interoperability experiments so that technical data can be gathered for detailed analysis.

The mission of the TDC is accomplished by executing the following functions:

(1) Evaluate candidate Army tactical data systems and equipment to determine the merit of incorporation into the field Army.

(2) Identify, test, and evaluate telecommunications support required by the Army.

(3) Perform tests to provide objective measurements of performance against prescribed standards and assess military worth and technical suitability.

(4) Evaluate hardware and software configurations as changes in (or additions to) user requirements occur and/or advances in the state-of-the-art make future generation ADP equipment feasible.

(5) Investigate the technical implementation alternatives to satisfy Army and Joint/International compatibility and interoperability requirements.

(6) Experiment with and analyze performance of new tactical ADP hardware and software.

(7) Analyze man-machine interface.

(8) Analyze currently acceptable performance thresholds and, if necessary, develop more appropriate ones.

(9) Develop and maintain a current comprehensive technical software and test library for TDC use.

(10) Evaluate prototypes of non-standard equipment and software.

It is our intention to accomplish the TDC mission by being a laboratory type facility that will have the capability to test and evaluate technical concepts, systems integration, man-machine relationships, and to confirm communications interoperability by "hands-on" experimentation and demonstration with innovative hardware and software techniques.

Approach:

The development of sophisticated ADP equipment is expensive in the commercial world and even more so in the military one. The severe military environmental requirements place an additional burden on the developer which is reflected in the system costs. The high costs result in a minimum set of equipments being built during the engineering development phase; consequently, this results in tight scheduling, with all the contractors and agencies involved in the development not having enough equipment for as long as they require it. The TDC recognized this problem and is minimizing its effects by emulating the equipments not readily available.

In December 1972, the Army let a competitive procurement for a Microprogrammable Multi-Processor (MMP) System that would have the capability of simultaneously emulating up to three tactical data systems. The systems would have the capability of intercommunication, and the MMP the capability of doing performance monitoring and reducing the data. The procurement resulted in a contract being awarded in May 1973 for the MMP System depicted in Figure 1.

THE MICROPROGRAMMABLE MULTI-PROCESSOR SYSTEM

The MMP System block diagram is shown in Figure 2. Seven processors are included in the existing MMP system. Five of the processors (5605) have 32-bit data widths, while the remaining 2 processors (5601) have 16-bit data paths. The fundamental architecture of the 5605 and 5601 is the same except for the size of the data paths and the number of interrupts and status registers. There are currently 5 Large Plane Memories (LPM) in the system which are designated 5603. The 65K words per memory bank provide a system capacity of 327,680 words. The basic cycle time is 950 nanoseconds with a maximum cycle time of 1.1 microseconds when connected with maximum cable length in the multiport configuration. The 8 ports on each memory allow each processor to simultaneously access a different memory bank. Priority is on a first come/first serve basis with the scanner mechanism working on a rotating basis so that no processor gets locked out.

One 32-bit processor (5605) performs the overall system control function (MP-60), and the remaining 3 are used for independent system emulations. (The fifth 32-bit processor is used for disk I/O.) The 5605 is housed in a ruggedized 80-card enclosure and is partitioned to allow implementation in word lengths of from 8 to 32 bits in 8-bit increments. The micromemory consists of 4096 microwords of 32-bit length. The micro-architecture has an eight field microinstruction with 80 nanosecond cycle time. The microarchitecture accepts both read/write (R/W) and read/only (R/O) micromemory.

FIGURE 1. THE MMP SYSTEM

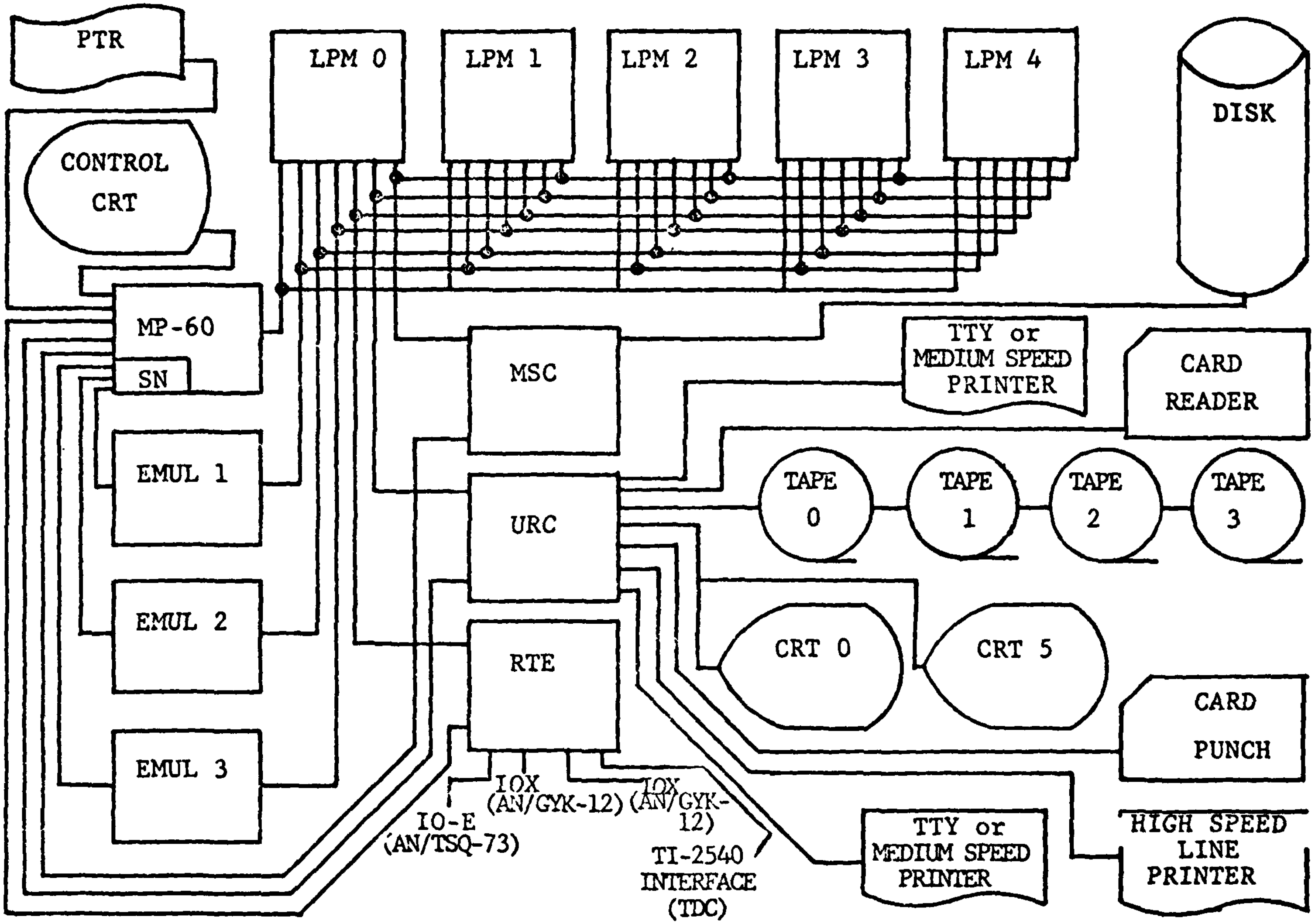


Figure 2. MMP Computer System

The MMP system utilizes R/W micromemory because of the requirement to run emulations of many different computers. The read/write micromemory can be loaded from paper tape, disc, or memory.

The remaining 3 processors are used for system input/output (I/O). The 32-bit I/O processor, Mass Storage Controller (MSC), is used to control the 844-2 mass storage disk. The disk has a capacity of 869 million bits with a transfer rate of 6.8 MHz and a latency time of 8.33 milliseconds.

One of the remaining I/O processors, designated the Unit Record Controller (URC), is used to control the standard ADP peripherals which include an 1150 line/minute 136 column line printer, 1200 card/minute reader, 250 card/minute punch, 400 character/second paper tape reader, four 800 BPI magnetic tapes, 2 low speed printers, and 7 alphanumeric displays.

The last I/O processor, designated the Real Time Equipment (RTE) is unique in that it allows actual peripheral equipment to be used in the system when available. The RTE currently has interfaces for the TACFIRE and AN/TSQ-73 systems and an NTDS interface which permits it to be driven by another computer within the TDC.

In performing emulations, a significant amount of execution time and micromemory are taken up in deciphering (cracking) the operation code of the target machine so that the host machine can perform the proper machine operation. A completely generalized emulator would perform this operation through a sequence of shifts, masks, compares and gates, and, depending on the complexity of the target machine's operations code and its similarity to the host machine, the emulation can result in large usage of microcode and correspondingly long execution times. The transform modules, which are an integral part of the emulator hardware, improve the emulator efficiency in terms of speed, instruction manipulation, and usage of micromemory. The basic function of the transform is to allow efficient cracking of the operation code of the target machine. It accomplishes this by gating fields of the instruction being emulated into selected registers of the host machine where microsubroutines can be implemented to perform the function to be emulated. Some transforms are so general that they are used in many tactical emulators while others are obviously tailored to a peculiar architecture. These transforms have been implemented with special purpose logic boards on the MMP system, but the latest technology utilizes Field Programmable Logic Arrays (FPLA)which are more flexible and cost effective.

Software/Firmware:

The MMP software consists of conventional software normally provided with a system of this size and special software that permits the generation of emulators, control of a multi-processor system, and performance monitoring capability. The real time operating system, MPX/RT, executes in the controlling CPU and is written in MP-60 assembly language. MP-60 is a pseudo architecture that was developed by Control Data Corporation for the 5605 series machines. All 5605 machines are completely interchangeable so that at any one time, any one of the 4 machines can act as MP-60. The operating system, MPX/RT, simultaneously supports both background and foreground jobs where the foreground job involves control of the emulations running and the background job is for batch processing, such as data reduction of previously run experiments. System Nucleus (SN) is an extension of the real time operating system that can simultaneously control the emulation of three independent systems. Typical functions performed by SN are the mapping of tactical peripherals onto simulated peripherals, recording of time of events, coordination among emulated systems, and allocation of overall system resources.

In addition to the operating system, the MMP has a COMPASS assembler, FORTRAN compiler, source program maintenance (COSY), object program maintenance (PRELIB), mass storage file maintenance (OCARM), relocatable loader, program checkout utility, System Nucleus utilities, and a microassembler. The microassembler is a unique tool that permits the generation of emulators. It provides listing control, conditional assembly, cross reference listing, multi-assemblies, and diagnostics.

Performance Monitoring:

A real time tactical data system has to have reserve processing capacity to allow the system to respond in a timely manner to a stress condition. The allocation of internal priorities in a tactical ADP system needs to be analyzed to prevent system lockups, or the case where a low priority task with very small processing time not being serviced because a higher priority job with large processing times always being in the queue before it. It may well benefit the system designer to develop a strategy that would prevent this from happening by permitting the lower priority job to interrupt the more important task after some preset time in the queue. Unfortunately, to even know the condition exists is a challenge in that it is very difficult to obtain correlative data in tactical data systems because of the lack of performance monitors and scenario drivers. Performance monitors used on commercial systems are either software (and part of large operating systems) or hardware with literally hundreds of probes used to monitor key registers and data paths within the machine being measured.

The military construction of computers (e.g., shielding) does not always allow all points of interest to be accessed by the hardware monitor.

The advent of medium and large scale integration (MSI and LSI) has compounded this condition bwcause the interconnection of registers that contain data, addresses, and machine states are done on the same chip and do not provide access to the outside world. The software monitors are typically embedded in the operating system and usually rely on recurring events so that sampling theorem can be used to analyze operating conditions. This technique has been used effectively for many years to optimize batch systems. The main intent is to determine where the bottlenecks are within a system so that allocation of resources can be optimized for a maximum return on investment. Insertion of the software monitor contributes to overall execution time and takes core and secondary memory space. Tactical data systems typically have asynchronous inputs, limited storage, and require fast response. Some of the key parameters to be measured such as interrupt response time, channel utilization, time in queues, etc., would be distorted by the software monitor.

The MMP system achieves this goal by embedding the performance monitoring capability in the microcode so that object code runs unaltered on the emulated system. Breakpoints, traces, and instruction count within program levels are examples of data taken without interfering with normal program execution. Each emulator keeps track of its own CPU performance monitoring data. System Nucleus, which is an extension of the operating system, keeps track of input/output utilization and periodically collects data from each of the emulators for file recording on disk or tape. This type of collection is typically done on a predetermined basis such as when there is a change of a program level, when a performance monitor register is at some predetermined percentage of its capacity, or when a unique event has occurred on the target system such as an I/O event, overflow, or a certain part of the object program being executed.

THE TACFIRE EMULATION

The TACFIRE system is a tactical ADP system that provides flexible fire support for combined arms operation. The principal functions of TACFIRE are tactical and technical fire control. This includes evaluating targets, selecting units to fire, munitions to be used, and computation of firing data. The TACFIRE Battalion was the first system to be emulated on the MMP. The TACFIRE Battalion system that was emulated was comprised of an AN/GYK-12 CPU, IOU, two 8K 32-bit word memories, 131K 32-bit word memory (MCMU), artillery control console, two magnetic drums (RAM), and a removable magnetic tape cartridge. The MMP system allows the mix of simulated and actual peripherals, but, since there were no tactical peripherals available, all the peripherals were simulated on the MMP system. The AN/GYK-12 CPU was emulated on emulator 1 under the control of MP-60. The AN/GYK-12 processor has the following characteristics:

(1) 32-bit instruction word.
(2) One, eight, sixteen, thirty-two, or sixty-four bit data word.
(3) Cycle time of 2.2 microseconds for the two 8K memories.
(4) Cycle time of 2.8 microseconds for the MCMU.
(5) Memory access control and protection for program and I/O separate and internal parity checking.
(6) One-hundred basic instructions and fifty extended instructions.
(7) Nine addressing modes.
(8) Sixty-four program levels.
(9) Sixteen 32-bit general purpose registers and 16 page registers per program level and special purpose registers.
(10) Up to 126 I/O devices, each with program initiation but independently operated. Queuing for each program level provides stacking of interrupts and high speed multiprogramming switching.

The mapping of the TACFIRE system is shown in Figure 3. The AN/GYK-12 CPU is emulated on a 5605 (emulator 1). The microcode to perform the emulation, which included performance monitoring, consisted of approximately 3700 32-bit microwords. This included some instructions that are not currently part of the AN/GYK-12 instruction repertoire, such as floating point. The IOU function is mapped into System Nucleus (SN) where the translation is done between the codes used for the tactical peripherals and the standard instructions used for MMP peripherals. Memory management is handled through requests to SN which provide the emulator with page register assignment to translate TACFIRE MCMU to MMP memory. In the original MMP system, there were only 162K words of memory and, consequently, the 131K MCMU could not be mapped completely into MMP memory. SN handled memory management by swapping pages of MCMU between core and disk as they were needed by the emulator. When a page was not available in core, a page fault would be set in the emulator which would cause SN to initiate a page swap.

The TACFIRE artillery control console (ACC) consists of a receive display (RD), compose and edit display (CED), and a switch assembly. SN maps the three functions onto the alphanumeric display which permits both the CED and the RD to be displayed on the same screen. The indicator lights of the switch assembly are mapped onto unused portions of the CRT while the control keys of the alphanumeric display are used to implement the switch assembly functions.

In a similar manner, TACFIRE drums are mapped onto the system disk while the cartridge tape unit is mapped onto the system tapes. Devices that have no functional counterpart within the MMP system, such as the 4-foot by 4-foot Digital Plotter Map, are mapped onto the disk where the output data can later be dumped and analyzed.

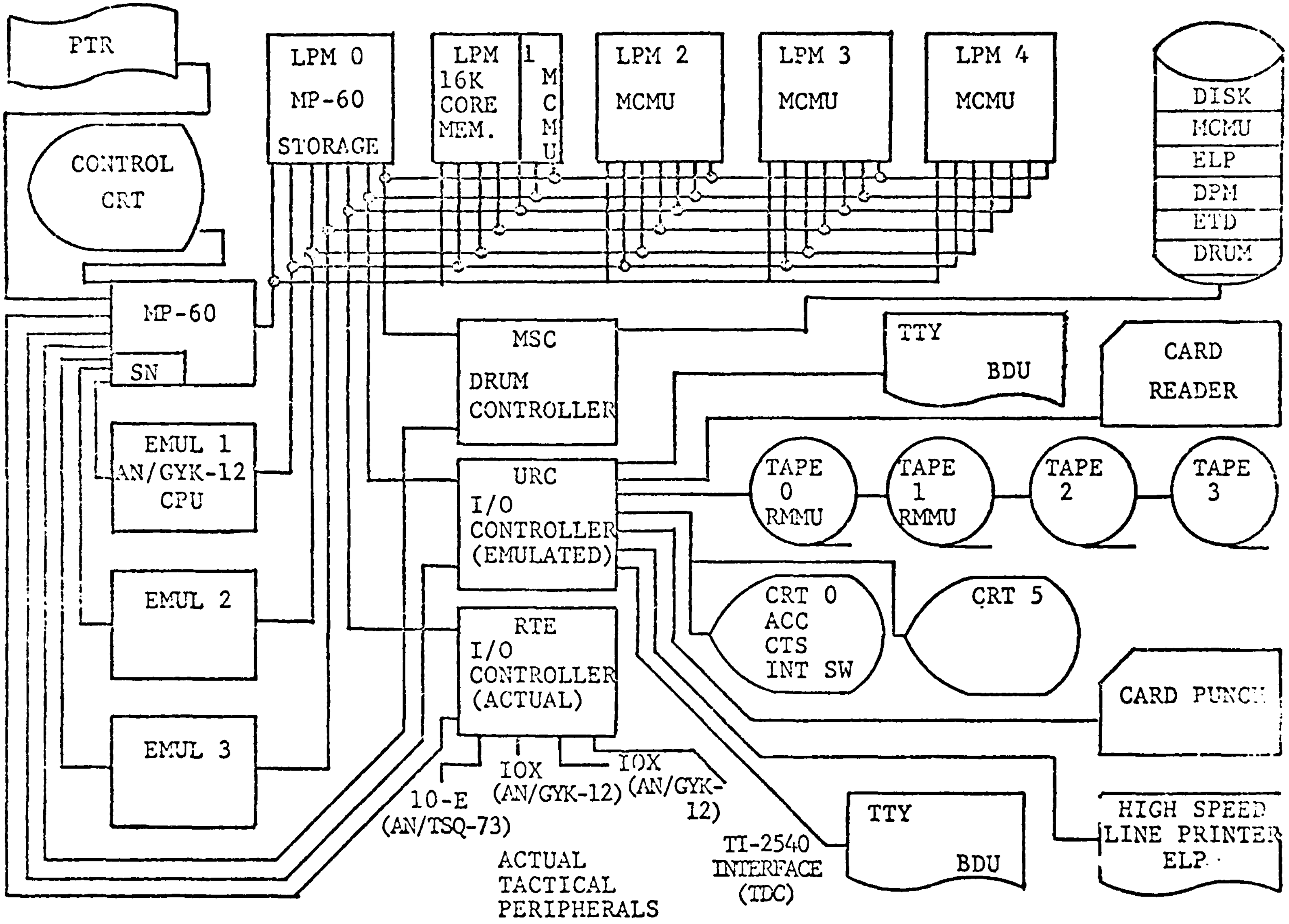


Figure 3. MMP Emulating the TACFIRE System

The TACFIRE system emulation behaves exactly as the actual TACFIRE. The application software is executed directly with no modifications. One difference that results, due to the lack of having actual tactical peripherals and the taking of performance monitoring data, is the difference in execution time. To achieve the objective of gathering performance monitoring data on program execution time, the emulator uses a translatable real-time clock called the virtual clock. The AN/GYK-12 is a hard-wired machine with an internal cycle of 125 nanoseconds. The emulator has implemented 29 counters, each corresponding to the duration of an internal AN/GYK-12 state. As stated earlier, the performance monitoring data can be collected at a prescribed time or when the counters get to some percentage of their total capacity. The integrity of the peripheral equipment execution time is performed by SN where the characteristics of both the MMP peripherals and the tactical peripherals are compared, and plus and minus delta times are computed.

A complex scenario was used to exercise the emulator. It was an adaptation of the same scenario used in the development and validation of the TACFIRE system. The scenario consisted of 8 firing units, fifty-five targets, and applicable geometry. The scenario was run and the same results were obtained as on the actual system.

The performance monitoring features embedded in the microcode of the MMP system are one of the key attributes of the emulated system. The use of microcode probes permit system performance parameters to be measured, completely transparent to the running of the application programs.

The system just described represented the DT/OT II TACFIRE configuration. Since that time, the magnetic drums and 8K core memories have been replaced by three 131K words of core memory at Battalion and four 131K words of core memory at Division. While this was a major change for the Project Manager for the actual hardware, it was accomplished on the MMP with minimum effort.

FEASIBILITY DEMONSTRATION

The success of the laboratory emulation prompted the Army to initiate an experiment to determine if an emulator could replace an actual tactical computer with no change in software, hardware, or degradation in mission response time. The cost of developing a tactical computer is a very small part of the total life cycle cost. Development of system software, doctrine, maintenance philosophy, training, and testing so out-shadow the cost of the basic computer that it became very attractive to determine if there could be injection of technology in the computer area with no impact on software. This would allow the Army to benefit in areas such as reliability, availability, maintainability, weight, power, and all the other parameters that are associated with the latest technology.

The TACFIRE Battalion with the AN/GYK-12 computer was used to obtain baseline mission execution times on various maintenance and diagnostic (M&D) routines and, subsequently, a complex scenario. The actual AN/GYK-12 computer was then disconnected and the emulator inserted; then, the same set of M&D's and the scenario were rerun. The results proved that the emulator could drive the peripherals and run the software unaltered at an approximately 20% faster rate.

INTEROPERABILITY

Interoperability within the Army is the ability of one system to receive and process intelligible information transmitted by another system. Interoperability includes communications, message structure and format, operational processing, operational procedures, and function requirements. In previous ADP systems tested, the normal hardware and software problems that are associated with system development were uncovered and fixed, but because of differences in multiple system philosophy, there were additional system deficiencies that resulted from operational procedures and man/machine interface.

The Army has many ADP systems either under development or proposed for the near future. Interoperability among these systems is currently undergoing intensive investigation since it is obvious that a human being will not be able to receive, digest, and forward the vast stores of information that will be acquired by these systems. During interoperability of some of the systems that have been tested to date, the tests had to be halted because of software problems, and the developer had to go into a "find-and-fix" mode. The other systems stand idle for this amount of time, and when system 1 is fixed, a similar cycle results when the other system develops faults.

Since emulators permit the execution of the target machine software without modification, the developer has the opportunity to run/debug software being written while the actual hardware is being used elsewhere (e.g., environmental testing), and also allows for software to be run <u>prior to</u> the actual hardware being built. One point that should be stressed is that emulation will not negate the need for final testing, but it should shorten the testing cycle that we are currently encountering due to undiscovered bugs in the software. Running the actual software on the emulated system with its performance monitoring capability will allow data to be taken and analyzed, patches to be inserted, so that a major portion of the system deficiencies will be uncovered and corrected prior to the investment involved with actual field tests.

The Teleprocessing Design Center is currently engaged in an experiment to assist in the early testing and validation of two Army systems. The two systems of interest are the TACFIRE DivArty and the Tactical Operating System Operable Segment (TOS^2).

The TACFIRE DivArty system is very similar to the TACFIRE Battalion described previously except for the magnetic drums and dual 8K memories being eliminated and 4 MCMU's (131K/MCMU) being inserted in their place. The TOS^2 system uses the AN/GYK-12 computer and most of the same peripherals as the TACFIRE system. The major differences are in the use of 4 militarized tape units and 4 magnetic drums. Additional Army peripherals to be simulated for the TOS were the Message Input/Output Device (MIOD) and Variable Format Message Entry Device (VFMED). The MMP system was expanded recently to include two low speed printers and five additional alphanumeric displays since the additional equipments were needed to model the new tactical peripherals.

CONCLUSION

This paper describes the MMP hardware and software past emulation accomplishments, current systems being emulated, and future plans. The MMP system provides the Army with the try-before-buy, use-before-delivery capability and allows rapid reconfiguration of existing equipment to permit the emulation of many different tactical computer systems. This quick assembly and performance monitoring capability allows interoperability experiments to be performed and software and procedures to be verified in a cost-effective manner. The capability of executing different computer instruction sets unaltered will permit software development testing, post deployment correction and enhancement, and research experimentation.

Interfacing CS^3 Facilities to the ARPANET

Marvin Schwartz

CENTACS

The Army has a requirement to achieve interoperability of its systems, both intra-Army and with other services and defense agency systems (JINTACCS Program). The Communications Research & Development Command (CORADCOM) Teleprocessing Design Center (TDC) at Fort Monmouth has proven that emulation is a viable tool to validate the performance of computerized systems.

The Army plans to use the TDC to support Army interoperability validation. However, there may be cases where emulation of systems or interoperable sets may not be practical, desirable, or even feasible. To connect or utilize actual tactical equipment for interoperability testing, a method of connection to the TDC is necessary. The Army plans to connect remotely located tactical systems to the TDC via the ARPANET to permit the interconnection of emulated and actual tactical systems.

Toward this end, the Army is developing a special general purpose interface unit that will provide hardware, software, and protocol compatibility between tactical systems (real or emulated) and the ARPANET's communication computer, the Interface Message Processor (IMP). In addition, remote users will be able to develop, test, and check out their software on the TDC's emulation system.

INTERFACING C^3 FACILITIES TO THE ARPANET

Marvin Schwartz

US Army Communications Research
and Development Command
Center for Tactical Computer Systems
Systems Validation Division
Fort Monmouth, New Jersey

INTRODUCTION

The Army has a requirement to achieve interoperability of its systems, both intra-Army and with other services and defense agency systems (JINTACCS program). To accomplish this objective the Army plans to use the Teleprocessing Design Center (TDC) to support Army interoperability testing and validation. The Communications Research and Development Command (CORADCOM) TDC at Fort Monmouth has proven that emulation is a viable tool to validate and test the performance of computerized systems. In addition to saving time and cost by emulation prior to actual tactical system testing, emulation provides system validators with the capability to perform a wide range of testing, coupled with performance monitoring, to demonstrate system operation and interoperability.

However, there may be cases where emulation of systems or interoperable sets may not be advantageous, desirable, or even feasible. The JINTACCS study report (Ref. 1) has determined that the best validation method is a combination of emulation, simulation, and use of actual tactical equipment for each system and interoperable set. To connect or utilize actual tactical equipment for interoperability testing, a method of connection to the TDC is necessary. This paper will describe how CENTACS plans to interconnect Fort Monmouth's TDC and other tactical facilities via the ARPANET.

In order to develop an optimum approach and utilize available hardware and software, a study effort was performed in conjunction with Control Data Corporation. The results of the study effort is documented in Ref. 2.

The approach taken to this effort is shown in Figure 1 and will be followed in this paper also. The desired capabilities generally fall into three functional classes. These are:

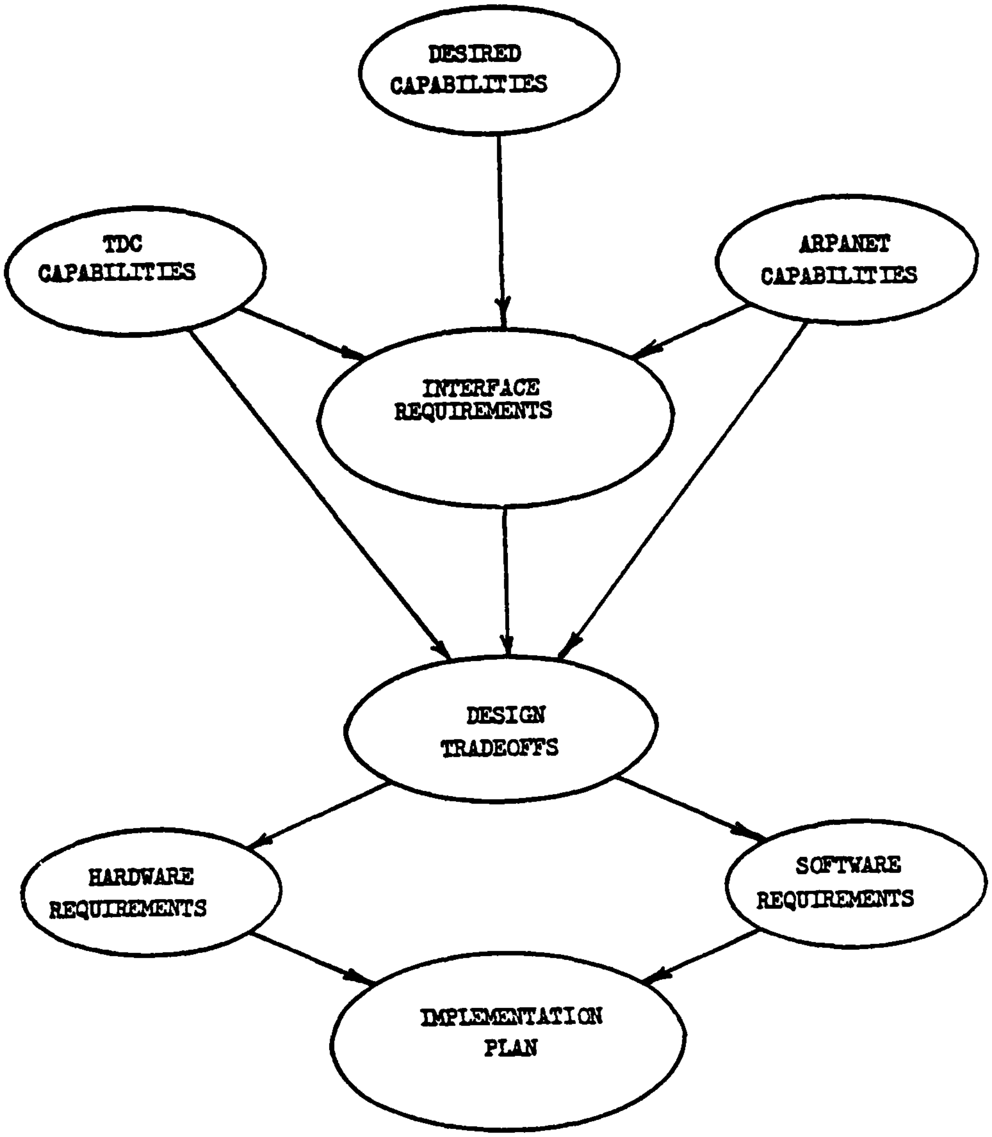


FIGURE 1. ARPANET STUDY APPROACH

a. utilizing the ARPANET for interoperability testing;
b. utilizing the ARPANET as a communication link to allow remote users access to the emulation resources of the TDC;
c. utilizing the resources available on the ARPANET.

ARPANET SYSTEM

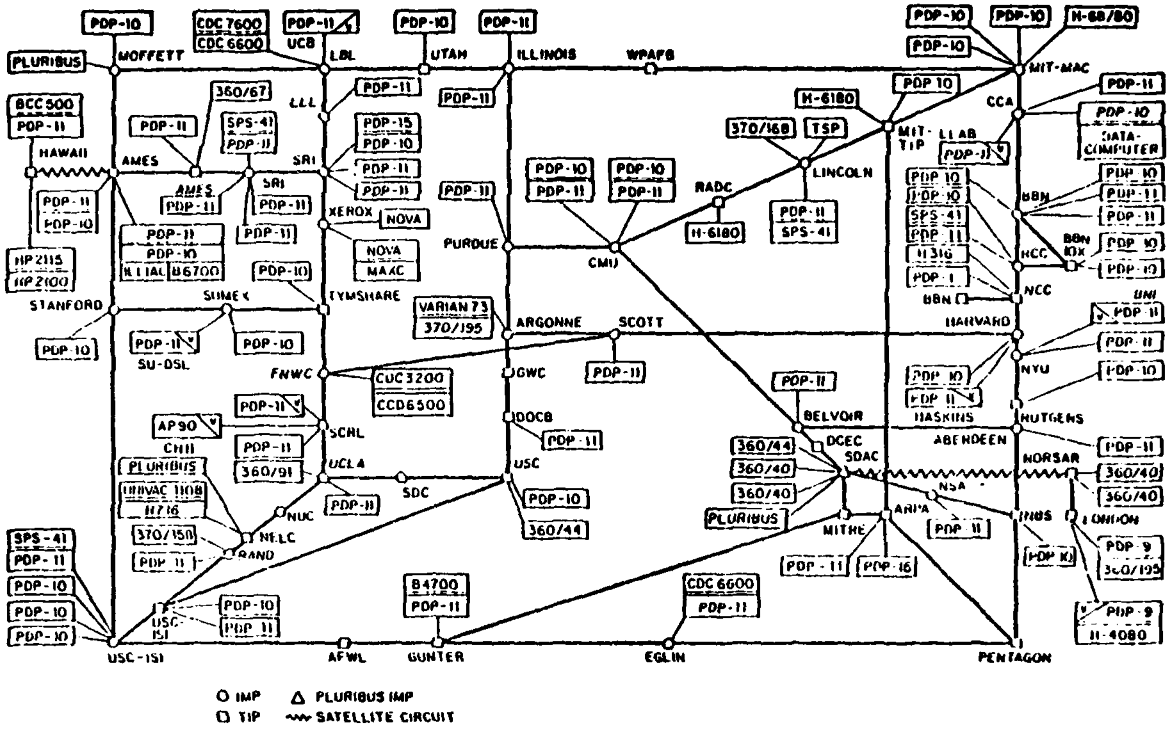
The ARPANET is a real time communication system. It is an operational, resource sharing inter-computer network linking a wide variety of computers. The network was designed to provide efficient communications between different types of computers so that hardware, software, and data resources could be conveniently and economically shared by a wide community of users.

The ARPANET (Fig. 2) originated as a purely experimental network in late 1969 under a research and development program sponsored by DARPA. Today the ARPANET connects computers in the United States, Hawaii, and Europe. Hawaii and Europe are connected via satellite to the United States. Entry into the ARPANET is via packet switching computers known as Interface Message Processor (IMP) or Terminal Interface Processor (TIP). The TIP is basically an IMP with a multi-line controller to permit terminals to enter the ARPANET and communicate with host computers.

IMP's and TIP's (Fig. 3) are tied together via dedicated 50 KBPS lines supplied by common carriers. Each node (IMP or TIP) is programmed to store and forward messages to the neighboring nodes in the network. Messages are sent from a terminal to its adjacent IMP or TIP. The message is forwarded through the network from IMP/TIP to IMP/TIP until it reaches the destination IMP/TIP where it is decoded and forwarded to the destination Host computer.

A packet network may be thought of as merely a long distance extension of a front-end processor. Data enters the network via a multiplexer (multiline controller). The multiplexer then transmits the bit stream to a message processor which breaks the data up into short blocks, or packets of data. The message processor adds a destination and origin address and other network information to each packet and interleaves the packets with blocks from other processors. This permits the most efficient packing of data onto the expensive long-distance circuits that make up the packet net's link. Types of links over which data is transmitted are satellite, submarine, leased telephone, common carrier, etc.

At each minicomputer-based node (IMP or TIP), or network connection point, all packets passing through the node are examined for their address. Those packets destined for local host computers or terminals are removed from the packet stream and stripped of addressing, diagnostic, and other



(PLEASE NOTE THAT WHILE THIS MAP SHOWS THE HOST POPULATION OF THE NETWORK ACCORDING TO THE BEST INFORMATION OBTAINABLE, NO CLAIM CAN BE MADE FOR ITS ACCURACY)

FIGURE 2. ARPANET LOGICAL MAP

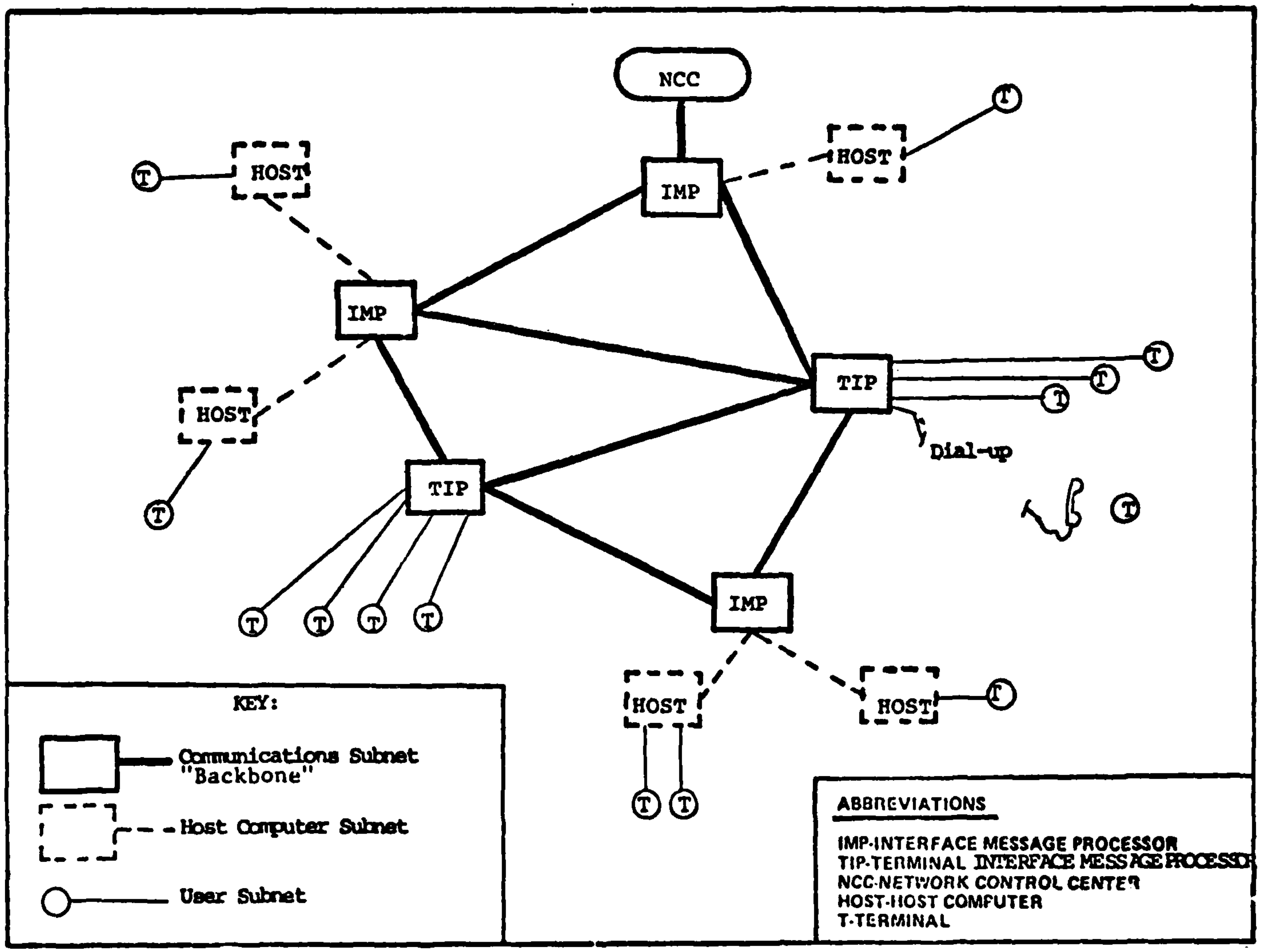


FIGURE 3. CONCEPTUAL DIAGRAM OF THE ARPANET

information added by the network. Then, after code translation, encryption, and other processing specified by the address protocols, the local packets are sent to their destination. All the other packets examined by the node minicomputer are retransmitted to the next node down the line via the most efficient path as calculated by an internal network protocol. At the next node the entire process is repeated for each packet.

What does this technique offer the computer user? As compared to conventional switched telephone or specialized data lines, the packet network sets up connections faster, can monitor errors and other transmission problems through diagnostics incorporated in the transmission software. Packet nets also facilitate conferences among interactive terminal users, program or file transfers between host computers, and other multiple or simultaneous data links. Packet nets avoid most of the problems of conventional switching nodes that create data errors on the telephone network. The path of each packet is not determined in advance, so different routes can be used to minimize traffic congestion and outages due to line failures, and packets are automatically checked at each node and retransmitted if in error or if receipt has not been properly acknowledged. All this takes but a fraction of a second and, if need be, is invisible to the user. Also, the code conversion features, not normally found on telephone or Telex-type networks, enable a wide range of dissimilar equipment to be interfaced without investing in specialized hardware or complex software routines. In short, a packet network can be "plug compatible" with any computer device.

DESIRED SYSTEM CAPABILITIES DESIGN REQUIREMENTS

The objective of making Fort Monmouth a node on the ARPANET is to provide the Army with the following capabilities:

a. transmit secure messages and data from tactical peripherals simulated on remote ARPA computers to the Microprogrammable Multi-Processor (MMP) system (a tactical emulation system) for interoperability testing (e.g., TACFIRE, TOS, etc.);

b. transmit as above from actual peripherals;

c. provide remote ARPANET users with tactical emulations developed on the MMP such as emulations of TOS, TACFIRE, AN/TSQ-73, etc.;

d. input scenarios from remote locations to drive MMP emulations via the ARPANET;

e. use ARPANET to link MMP (emulated tactical system) to other tactical computer systems (Navy, Air Force, etc). For example, JINTACCS joint interoperability testing could be performed by linking Navy tactical ADP systems to the Army's MMP;

f. transmit software updates via the ARPANET to tactical computers at remote areas. Software changes could be designed and checked out on the MMP;

g. transmit field problems to the MMP for diagnosis and correction;

h. an MMP emulation output could be used to drive a tactical peripheral located at a remote site;

i. Fort Monmouth will have the computing power of all the computer systems tied to the ARPANET;

j. the ARPANET will provide the capability to transmit written material (mail box);

k. the ARPANET connection will enable the MMP to be used as a remote training tool. For example, TACFIRE Artillery Control Console (ACC) operators at remote locations could interface with an emulated TACFIRE system.

The objectives of making Fort Monmouth a node on the ARPANET can be divided into two design categories:

a. establish the capability to transmit and receive data and messages over the ARPANET to and from tactical and commercial computer systems;

b. to provide ARPANET users with the computer resources of Fort Monmouth.

In the past two decades, numerous military systems have been conceived and/or developed with the objective of capitalizing on the latest technological developments. Typically, each system has been designed with specific functional and performance goals to solve a specific problem. Design optimization to meet these individual system goals has typically ignored the inevitable time when command and control requirements would dictate the exchange of information between systems.

As a result, each system design has its own special hardware, software, and data vocabulary. Figure 4 illustrates an example of a number of such systems which have been shown by previous analysis in Ref. 1 to require interoperability for exchange of information. A generalized analysis of the interoperability requirements has shown that design compatibility must exist at several levels of implementation. Figure 5 identifies six major levels at which design compatibility must exist in order to permit interoperability between systems.

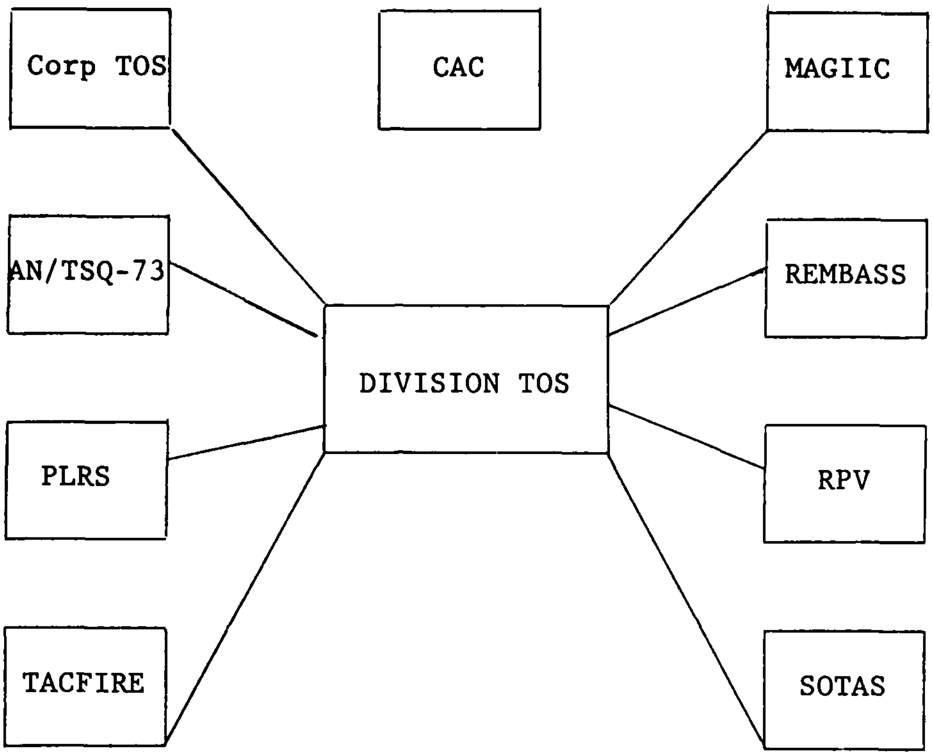


FIGURE 4.

SYSTEMS REQUIRING INTEROPERABILITY
FOR INFORMATION EXCHANGE

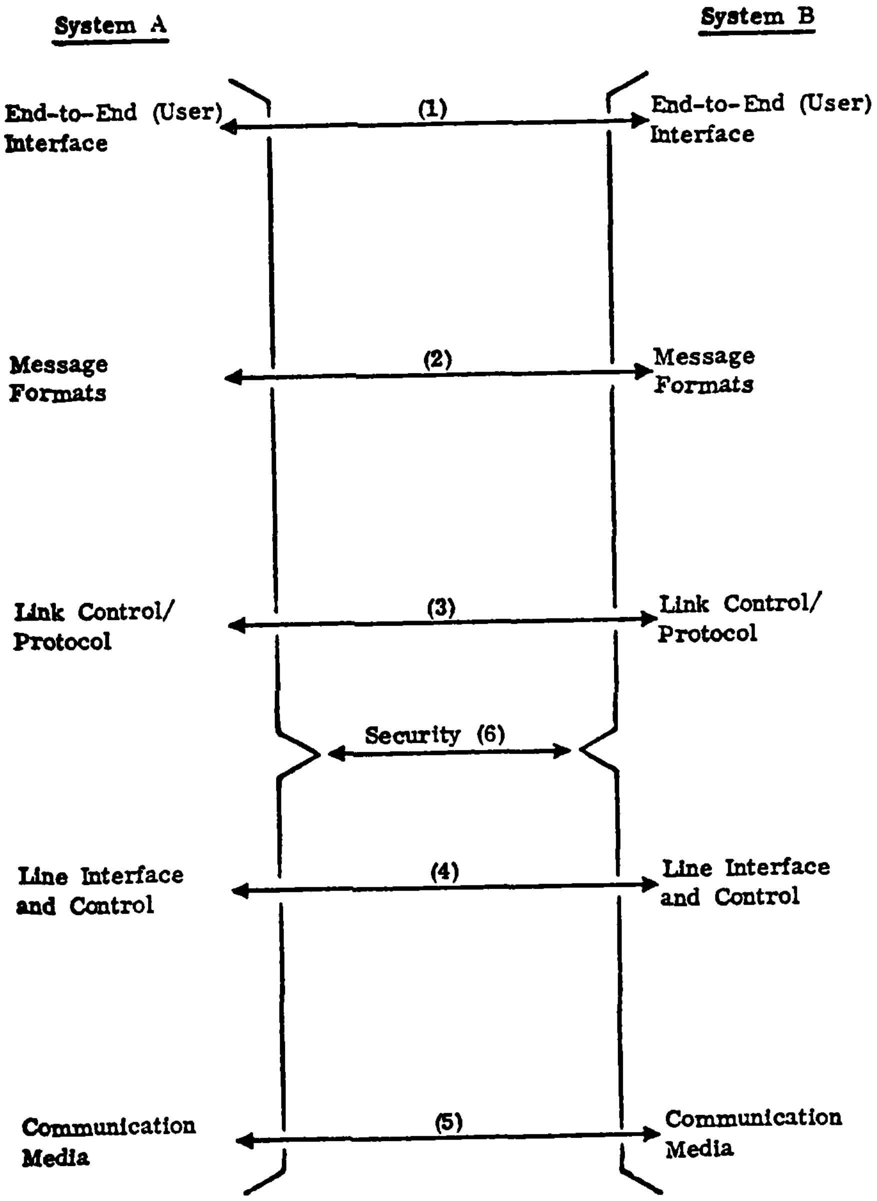


FIGURE 5. DESIGN COMPATIBILITY LEVELS BETWEEN SYSTEMS

The ARPANET will replace level 5 - the communication media. However, there are cases, as will be pointed out later, where the communication media affects level 3 (link control/protocol) and, in fact, dictates the design or procedure requirements for level 3. Level 4 (interface and control) will typically define the method of connection to the system and dictates the ARPANET connection requirements.

All exchange of information between the DivArty TACFIRE and the Bn TACFIRE occurs between the DivArty Fire Direction Center (FDC) and the Bn FDC, as shown in Fig. 6, using the artillery communications network. The artillery communications network provides for either wire, FM or AM/SSB radio, for the communications link. Each FDC is provided with the peripheral equipment necessary to ensure proper communications (Digital Data Terminal, Communications Control Unit, Remote Communications Monitor Unit).

The hardware required for the DivArty FDC to interoperate with the Bn FDC is identical to the communication equipment required within each system, since the systems were designed to interoperate. The hardware involved (identical at both FDC's) consists of the following devices: (1) COMSEC equipment KG-30/31; (2) Digital Data Terminal (DDT); (3) Communication Control Unit (CCU); (4) Remote Communications Monitor Unit (RCMU).

The choices for interfacing either a DivArty or Bn FDC to the ARPANET are as follows:

a. output of CCU;
b. output of DDT;
c. output of AN/GYK-12, IOX channel.

It will be shown that the IOX channel output is the only reasonable choice for the TDC interface to the ARPANET.

Investigations into driving remote tactical peripherals indicate that an interface to the Remote Data Terminal (RDT) is required. The RDT has characteristics equivalent to looking toward the CCU and DDT as a serial combination. Examples of peripherals designed to be remoted are:

a. Variable Format Message Entry Device (VFMED);
b. Battery Display Unit (BDU);
c. Digital Message Device (DMD);
d. Message Input/Output Device (MIOD);
e. Tactical Computer Terminal (TCT).

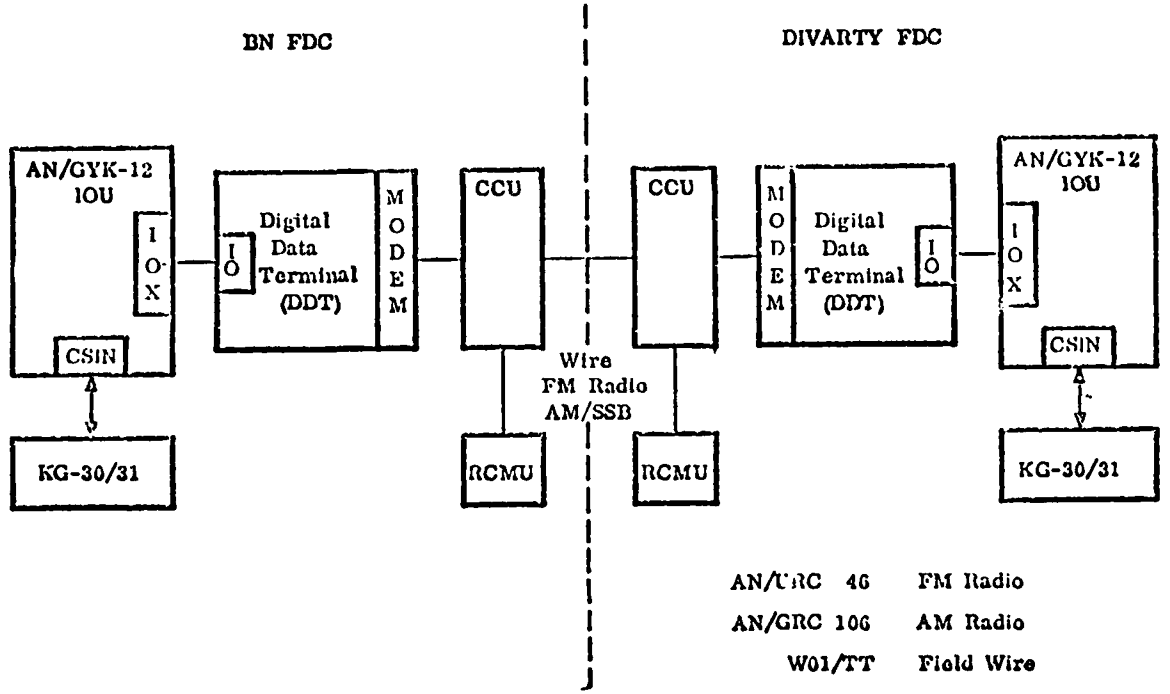


FIGURE 6. DIVARTY/BN INTEROPERABLE HARDWARE CONFIGURATION

The Army plans to develop a Universal Interface Box (UIB) which can interface both commercial and tactical computers and peripherals with only the I/O card being different. It was determined to interface one UIB to an IOX channel available on the TDC's MMP system and to interface the other UIB to an RDT interface. Therefore, this initial demonstration will verify the capability of both interfacing tactical computer (IOX channel) and the TDC's MMP and tactical peripherals (MIOD).

TELEPROCESSING DESIGN CENTER (TDC)

The Teleprocessing Design Center (TDC) Microprogrammable Multi-Processor (MMP), Fig. 7, provides a system which allows performance testing of tactical command and control systems by the use of emulation. The performance monitoring information becomes the basis for performance evaluation of tactical systems. The MMP computer system in the TDC has the necessary hardware, firmware, and software to emulate a total computer system.

The current configuration of the TDC consists of seven processors, 328K 32-bit words of memory, and associated peripheral devices as well as a Real Time Equipment interface. This system, together with its attendant software and firmware, is capable of executing three simultaneous emulations. An example of an emulation system currently in existence at the TDC is the TACFIRE emulation system (TES). TES provides a means of performance testing the Litton AN/GYK-12 computer, its application software, and the TACFIRE operating system (OS). The unique capabilities of the MP-60 computer system and the system nucleus allow a user to execute the TACFIRE software in a controlled environment and at the same time collect data at various points in the execution for later report processing. These reports are then used for evaluating the performance (e.g., core utilization, instruction usage, etc.) of the AN/GYK-12 computer and the TACFIRE OS.

The MMP system enables the laboratory to emulate other command and control computers. Emulation includes both the capability to execute the test system's software, producing the same results and the ability to translate the results into real time. Emulation under simulated battlefield conditions will provide insight into areas of possible data handling bottlenecks. The TDC can also emulate more than one system simultaneously, focusing on the ability of different tactical data systems to function with each other. Interoperability studies through emulation techniques, under laboratory conditions, will permit isolation of any one parameter for study.

The TDC, as currently configured, has a number of candidate interfaces to provide ARPANET access. In all cases, the interface from the TDC would connect with the Universal Interface Box (UIB). The interface candidate locations are:

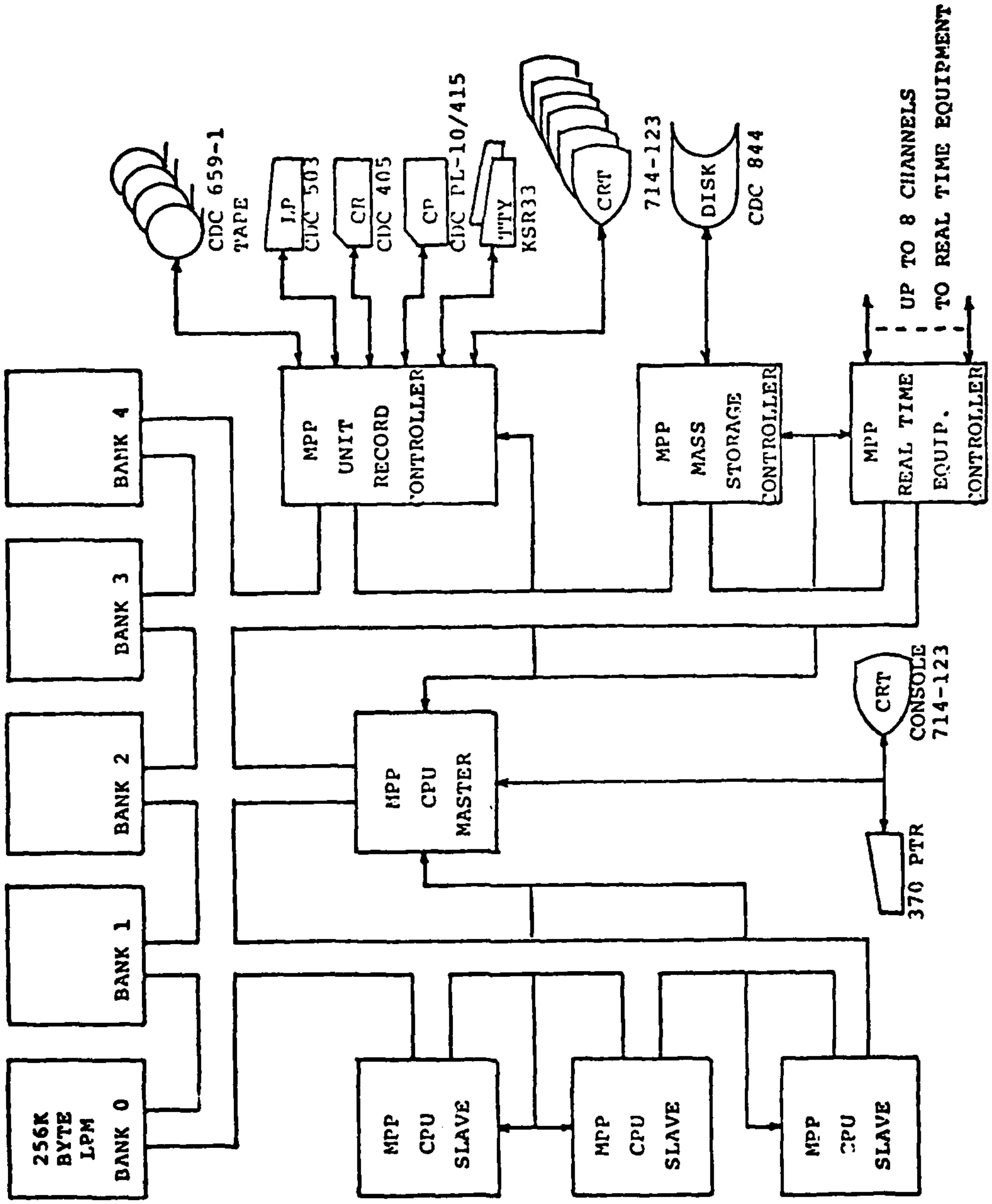


FIGURE 7. TELEPROCESSING DESIGN CENTER

a. the emulation system peripheral controller (URC);
b. a direct memory access (connected directly to the memory bus);
c. the Real Time Equipment Controller (RTE/IOX) IOX channel.

Connecting the UIB to the MMP in the TDC via the RTE's IOX channel offers the following advantages:

a. minimum changes to the MMP or tactical software;
b. the IOX interface can be used for both interfacing the MMP and the AN/GYK-12 computer (TACFIRE computer).

UNIVERSAL INTERFACE BOX - HARDWARE

In order to minimize development costs, existing hardware and software developments were looked at. Through past efforts in emulating tactical systems (TDC) and ARPANET implementations (PDP), a base of software and hardware have been developed that could be used for ARPANET interfacing. In addition, related studies and development efforts have resulted in some hardware interface designs that can be used which will result in significant development savings. Three alternatives exist that will, in varying degrees, take advantage of existing hardware and/or software to provide the various interfaces and the processing capability required for interoperability testing. These are:

a. MP-60 development;
b. MMP Emulator;
c. PDP family hardware.

Fig. 8 contains the advantages and disadvantages of utilizing the three different types of computers. Based upon a trade-off analysis, the PDP family computers appear to be the most logical and cost effective choice. The Transmission Control Program (TCP) was developed for the PDP-11 machines by DARPA. Furthermore, if and when the MMP system is expanded into a resource for other users to access the ARPANET, the PDP-11/34 minicomputer (the computer selected) could be easily expanded to a PDP-11/70 system which has interactive multiprocessing and multiprogramming capability. In addition, there exists interfaces between the DEC hardware and the ARPA IMP.

Within the PDP-11 family there exists software compatible computers ranging from a micro to a large time-sharing system. As a result of memory requirements, the PDP-11/34 was selected as the optimum sized computer for the UIB. With the PDP-11/34 the Army plans to procure a complete set of peripherals to develop software and monitor system operation and load the computer programs.

APPROACH	ADVANTAGES	DISADVANTAGES
PDP	1. CAN UTILIZE EXISTING TCP SOFTWARE WRITTEN FOR PDP-11/03 AND MAINTAINED BY ARPA AGENCY. 2. 1822 TO UNIBUS INTERFACE AVAILABLE. 3. RS232-C INTERFACE AVAILABLE. 4. MAY HAVE OTHER USEFUL APPLICATIONS, SUCH AS SPS.	1. PROCURE PDP MACHINE FOR TDC. 2. DEVELOP UNIBUS TO IOX INTERFACE. 3. DEVELOP NTDS INTERFACE. 4. DEVELOP INTERFACE TO REPLACE DIGITAL DATA TERMINAL (DDT). 5. DEVELOP CSIN INTERFACE.
EMULATOR	1. USE EXISTING MPP FOR EMULATION OF PDP INSTRUCTION SET. 2. NTDS INTERFACE AVAILABLE. 3. RS232-C INTERFACE AVAILABLE. 4. IOX INTERFACE AVAILABLE. 5. CAN UTILIZE EXISTING TCP SOFTWARE WRITTEN FOR PDP-11/03 AND MAINTAINED BY ARPA AGENCY. 6. CSIN INTERFACE AVAILABLE FOR KG-30/31.	1. DEVELOP FIRMWARE FOR PDP INSTRUCTION SET EMULATION. 2. POSSIBLE HARDWARE CHANGES REQUIRED FOR EMULATION. 3. DEVELOP 1822 INTERFACE. 4. MPP PROBABLY OFFERS EXCESS PERFORMANCE PER DOLLAR RATIO IN REMOTE APPLICATIONS. A PDP MACHINE COULD BE SUBSTITUTED SINCE SOFTWARE BASE IS PRESERVED.
USE MPP AS MP-60	1. USE OF EXISTING MPP. 2. NTDS INTERFACE AVAILABLE. 3. RS-232C INTERFACE AVAILABLE. 4. IOX INTERFACE AVAILABLE. 5. CSIN INTERFACE AVAILABLE FOR KG-30/31.	1. DEVELOP TCP AND SUPPORT SOFTWARE. 2. SOFTWARE CAN NO LONGER BE MAINTAINED BY ARPA AGENCY. 3. MP-60 PROBABLY OFFERS EXCESS PERFORMANCE PER DOLLAR RATIO IN REMOTE APPLICATIONS.

FIGURE 8. CPU COMPARISON

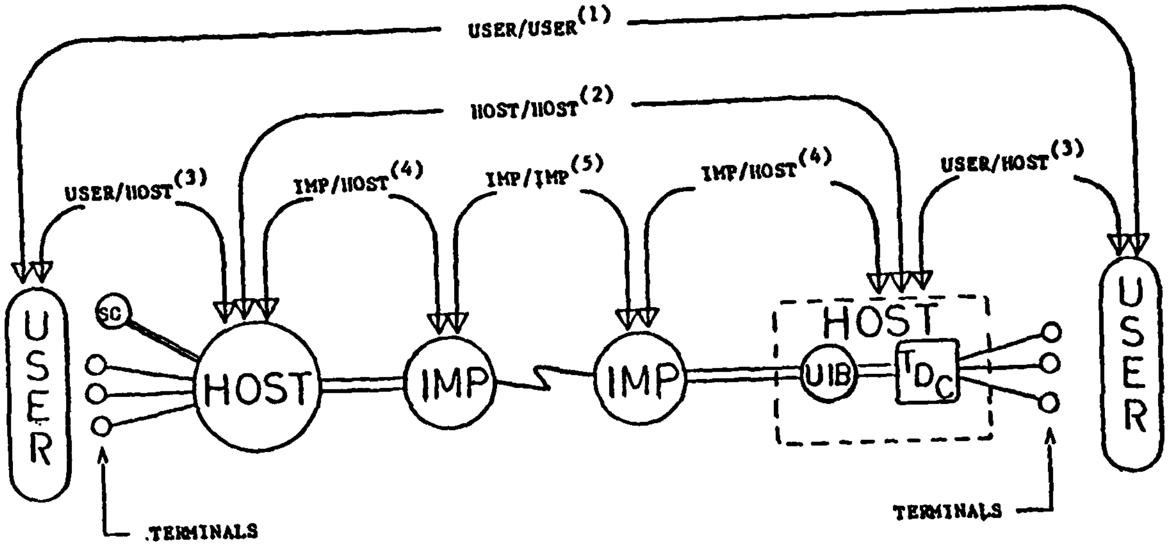
UNIVERSAL INTERFACE BOX - SOFTWARE

In the past, in each Host computer a program referred to as the Network Control Program (NCP) was implemented and added to the Operating System (OS) of the Host. This implementation is typically major surgery for most Hosts. The NCP allows Host computers to communicate with each other according to a Host-to-Host protocol, which is a network-wide standard. To minimize OS modifications and to develop a protocol compatible with packet radio, DARPA developed the Transmission Control Program (TCP). TCP is the latest version of the ARPANET protocol, which enhances the reliability of transmission. Although still in the development stage, TCP appears to be the wave-of-the-future and DARPA is recommending replacement of NCP with TCP for all ARPANET Hosts.

The ARPANET philosophy of communications consists of a layered protocol scheme as depicted in Fig. 9. This approach of defining a layered protocol scheme (in some cases implemented with both hardware and software) has the advantag? of segmenting communications interfaces into small hardware/software packages that can be individually implemented. In addition, changes can be made at either end of the spectrum without modifying the entire scheme. It also means that the level to which one carries his investigation of the ARPANET depends on his ultimate goals and requirements. Typically, one is not required to examine in detail all of the layers. Of main interest to us is the IMP/HOST protocol/interface, also known as the 1822 interface developed for DARPA by Bolt, Beranek, and Newman.

At the top end of the scale is the user-user protocol. The user-user protocol would allow a remote terminal user, for example, to utilize the resource provided by UCLA for an application involving a mathematical model calculation. One such protocol is called TELNET. TELNET provides for the establishment and control of network connections (User-Host), specifications of various operating options (User-User), and handles interactive data flow between the network and the user (User-Host/User-User).

The User-Hoct protocol defines the interface between the TELNET and the Transmission Control Program (TCP). The Host-Host protocol defines how the TCP at the source converses with TCP at the destination. The Host-IMP defines how control and data are transferred from the TCP in the Host or UIB to the IMP. The IMP-IMP protocol defines how messages (packets) are transmitted from the source to the destination.



NOTES:

(1) TELNET

(2) NCP

(3) LOCAL LINE

(4) 1822

(5) ARPANET

FIGURE 9. ARPANET LAYERED PROTOCOL SCHEME

The TCP program provides for a reliable, error free logical communications channel and ensures end-to-end acknowledgment, error correction, duplicate packet detection, packet sequencing, and packet flow control. Fig. 10 is a block diagram of the software modules which are planned to reside in the UIB. The OS provides for the following facilities:

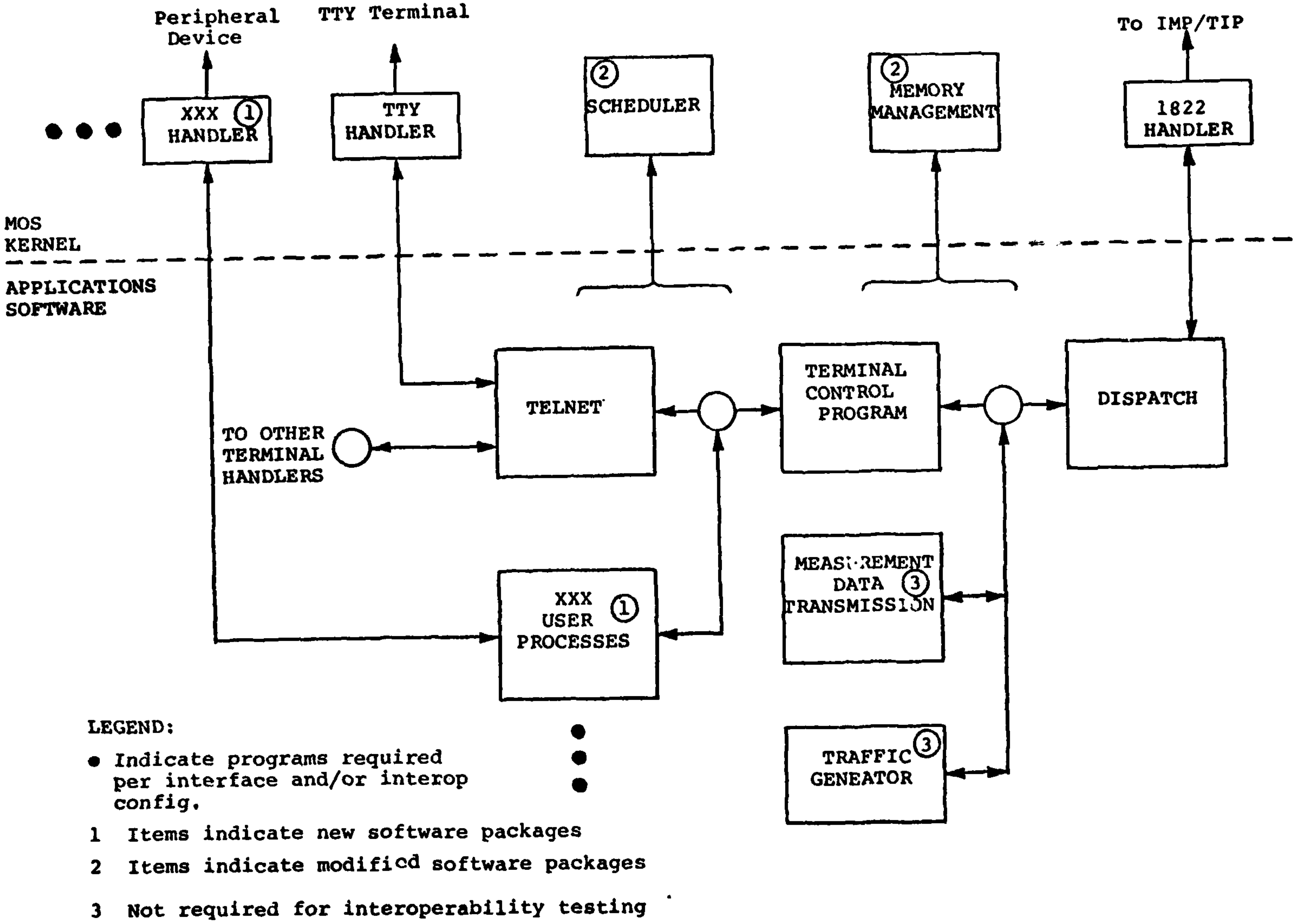
a. a multiprocess real-time environment;
b. an interprocess communication/synchronization mechanism;
c. sharing of peripheral devices;
d. asynchronous input/output facilities;
e. storage management primitive.

The dispatch program (DSP) performs the function of multiplexing packets received from the 1822 interface. For packet transmission, DSP merges the output from several processes into one stream to send out over the 1822 interface. For packet reception, DSP breaks up the stream received from the 1822 interface into several streams for proper distribution to the processes. Logical processes are defined to be TCP, Measurement, etc. Reliable reception/transmission of packets is accomplished using end-to-end acknowledges.

User processes and handler is the new software that would have to be developed as an interface to an IOX or RDT channel. In addition, an ARTCOM protocol for the simulation of traffic on the artillery communication network is planned to be developed. The ARTCOM protocol is based on transmission priority after the time in which the net is sensed to be clear. The simulation of this protocol is to have each UIB within the system capable of determining when the net is busy and capable of relaying this information to its users. The adopted network protocol keeps each UIB on the network aware of net busy conditions. Each UIB in turn has the capability of signaling a busy condition to its user, a computer or peripheral.

The software required to upgrade the TDC's MMP system to resource for users on the ARPANET was investigated. The major differences between interoperability testing software and software required to use the emulation and software development capabilities of the TDC occur when the TDC is required to handle the remotely submitted jobs and return them to the respective requestors. The ability to operate the TDC as an ARPANET resource does not exist in the TDC and must be developed. The changes required to support the TDC as a resource occur in the MP-60 software, emulation firmware, and the UIB software and hardware.

The MP-60 software must be changed to allow a job to be received, scheduled, executed, and returned to the UIB for transmission to the user. The firmware must be changed to allow for programmable "hooks" to be added to the



LEGEND:

- Indicate programs required per interface and/or interop config.

1 Items indicate new software packages

2 Items indicate modified software packages

3 Not required for interoperability testing

FIGURE 10. PROPOSED UIB SOFTWARE DIAGRAM

firmware to suit the users' testing requirements during emulation. The UIB must be changed to allow for access of the TDC's emulation and software development capabilities by remote users. As such, the UIB must queue all remote and/or local requests for data processing.

CONCLUSIONS AND IMPLEMENTATION PLAN

The desired system capabilities generally fall into three classes which lend themselves to a phased implementation approach. These are:

a. Phase 1 - using the resources of the ARPANET;
b. Phase 2 - using the ARPANET for interoperability testing;
c. Phase 3 - using the TDC as a resource via the ARPANET by remote users for software development and validation.

Phase 1 is being implemented by acquiring a TIP which will be installed at Fort Monmouth. The TIP will allow, via terminals, access to the existing ARPANET resources.

Phase 2 will be implemented by the development of two UIB's between the TIP and the tactical computer peripheral and MMP and the development of the necessary software to accomplish interoperability testing.

Phase 3 is planned to be implemented by upgrading the UIB to a time-shared processor and upgrading the MMP's MP-60 processor and firmware to allow time-shared users access to the emulations.

Figure 11 is the system configuration recommended for implementation and system verification/feasibility testing. Initially, it is planned to check out the complete system at Fort Monmouth and then the second UIB will be moved to a remote tactical facility and final end-to-end performance testing will be conducted.

If this approach proves to be a viable and practical means to communicate and perform interoperability testing among tactical facilities, additional units will be installed at tactical computer facilities throughout the country.

REFERENCES

1. ARMY JINTACCS INTEROPERABILITY REPORT prepared for Cdr, CORADCOM, Center for Systems Engineering and Integration, Fort Monmouth, New Jersey, under Contract No. DAAB07-77-C-3054, February 1978, by Control Data Corporation, Aerospace Division.

2. ARPANET STUDY - A Study to Investigate Interfacing the Teleprocessing Design Center (TDC) and various Military Computer Facilities Throughout the Country to the Advanced Research Projects Agency Communications Network (ARPANET) prepared for US Army Electronics Command, Fort Monmouth, New Jersey, under Contract No. DAAB07-78-C-3301, April 1978, by Control Data Corporation, Aerospace Division.

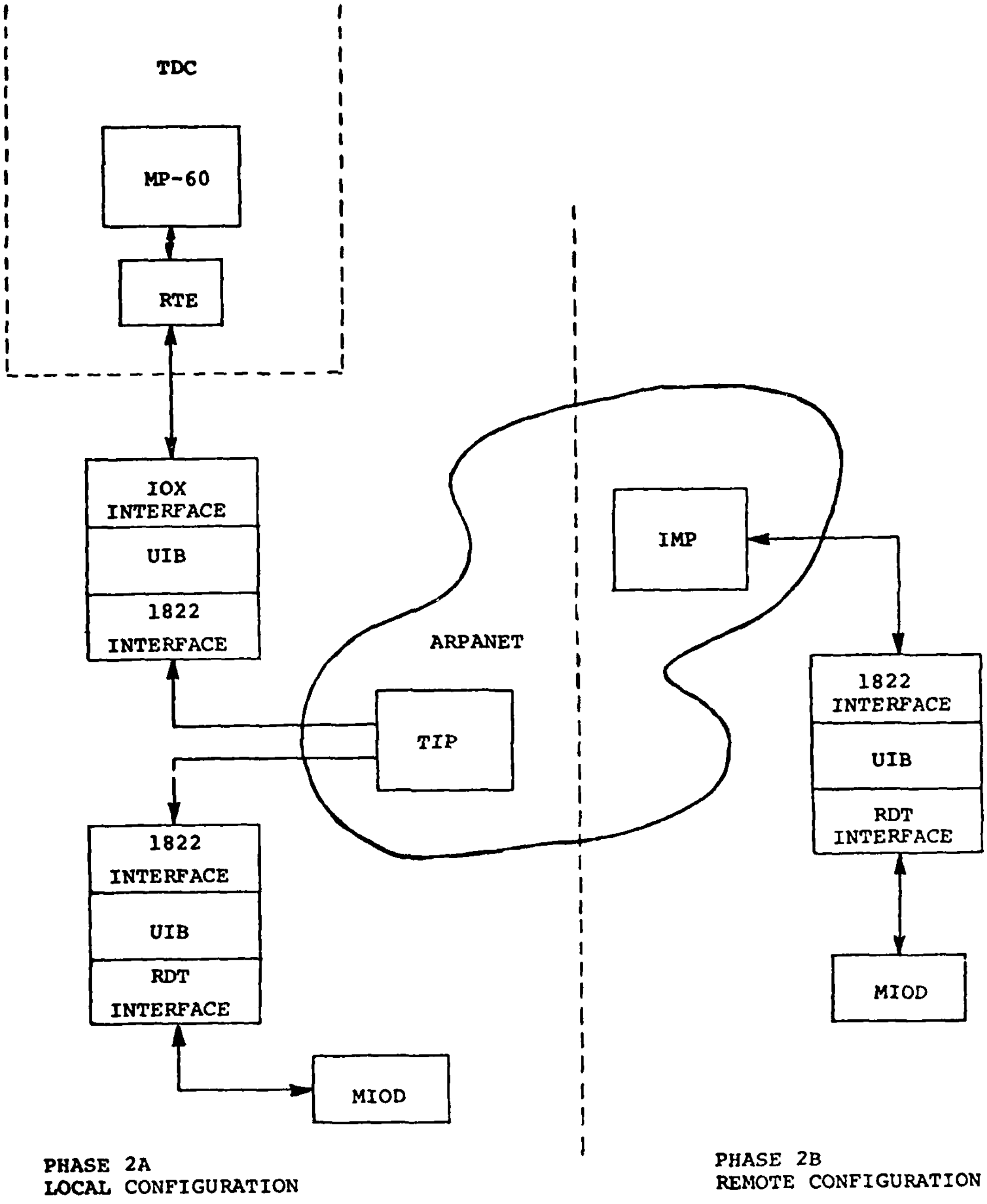


FIGURE 11. RECOMMENDED SYSTEM CONFIGURATION FOR PHASE 2

Software Development Support System (SDSS)

Bernard Newman
Roy Mattson

The Software Development Support System (SDSS) is a planned government-owned computing facility configured to satisfy the program development needs of users/contractors engaged in the Military Computer Family (MCF) demonstration model validation effort. SDSS is intended to provide a centrally located facility of commercial off-the-shelf hardware and software augmented by a variety of advanced, specialized software tools required to support the various life cycle development activities of militarized computer systems. SDSS will provide a host environment where such activities as configuration management, test, verification and validation, training, and post-deployment support can be conducted. SDSS will address the problems of uncontrolled tactical software development arising from non-government ownership of development software and the proliferation of development tools. SDSS will attempt to reconcile the conflicting requirements of support software developers and support software users.

SOFTWARE DEVELOPMENT SUPPORT SYSTEM (SDSS)

Bernard Newman
Roy Mattson

US Army Communications Research
and Development Command
Center for Tactical Computer Systems
Systems Validation Division
Fort Monmouth, New Jersey

INTRODUCTION

The prime impetus for the establishment of a centralized Software Development Support System (SDSS) under direct Government control and ownership arises from the myriad of difficulties which have plagued past and on-going tactical computer based software system developments. In particular, the initial SDSS configuration to be physically located at the Center for Tactical Computer Systems (CENTACS), Fort Monmouth, will be used to fully control the life cycle development of the Military Computer Family (MCF) demonstration system software. This will include the full complement of operational software, tools, and specialized software used to generate, develop, and test the target system software. SDSS will provide a single, centrally located PDP-11/70 host facility augmented by DEC Software, the Bell System Programmer Work Bench (PWB)/UNIX Operating System, and the DoD-1 compiler and language dependent support software tools. The facility will be linked to remote user/developers via dialed telephone lines and through the ARPA-net. SDSS will utilize commercially available hardware/software to the maximum extent possible and will be designed to provide a convenient working environment for programmers and MCF system developers consisting of a user community of about one-hundred people. (SDSS should be capable of handling 50-60 simultaneous on-line users.) The projected capabilities of SDSS are envisioned as growing and evolving at a rapid rate as accumulated experience provides support for the efficiency and effectiveness of its implementation. This paper presents the SDSS concept and describes the planned system configuration. Advantages which will accrue to the Government are also presented.

THE SDSS CONCEPT

The SDSS concept embodies a number of different ideas concerning life cycle development of tactical software. In general, tactical system program development and the execution of the resulting programs are two radically different functions. The selection of a DEC PDP-11/70 and

PWB/UNIX for SDSS is intended to provide a single uniform interface to both the developers of MCF applications software and the developers and maintainers of SDSS support software and tools. SDSS is intended to provide the mechanism whereby the standardization, control, and improvement of available support and target software subsystems, such as compilers,real-time operating systems, data base management systems, and command languages can be effected. Tactical software life cycle cost and acquisition time will be substantially reduced by the ready availability of such a time shared facility, where programmers have access to one or more computer systems equipped with a growing body of well-supported standardized support tools, software subsystems, and newly developed programs. Figure 1 provides a delineation of the major tactical software problems being addressed by SDSS.

- UNCONTROLLED TACTICAL SYSTEM OPERATIONAL SOFTWARE DEVELOPMENT
  - GOVERNMENT NON-OWNERSHIP OF DEVELOPMENT SOFTWARE
  - CONFLICTING REQUIREMENTS OF CONTRACTORS AND GOVERNMENT DEVELOPERS
- PROLIFERATION OF HOST ENVIRONMENTS AND DEVELOPMENT TOOLS
- DIFFERENT HOST AND TARGET SYSTEM REQUIREMENTS
- NON-AVAILABILITY OF THE MILITARIZED HARDWARE DURING EARLY SOFTWARE DEVELOPMENT PHASE

FIGURE 1. TACTICAL SOFTWARE DEVELOPMENT PROBLEMS

In particular, an analysis of tactical system software development case histories has shown that lack of support software visibility has ultimately resulted in significant problems when test, deployment, and maintenance phases of the development life cycle are entered. The consequences of ineffective control over support software results in the situation where Government personnel often find it difficult to generate, modify, understand, or enhance a particular operational/application software system without the designer's assistance. The Government is thus usually locked into particular contractors well beyond the development of initial prototypes. Many benefits associated with competition are lost in the areas of cost and quality when alterations, enhancements, and changes to the prototype systems are planned and subsequently implemented in the production system. In these matters, the objectives of contractors and government most usually diverge resulting in serious confrontations which adversely impact the particular tactical system's design evolution process. The non-ownership by Government of major development software components ultimately results in a proliferation of development tools and host environment facilities since the proprietary nature of this software often precludes its use by more than one contractor on more than one job. Since the requirements for a host environment are substantially different from that of the basic target system, an additional load factor of 30-40% of the basic system cost is usually built into each new development model. This is objectionable since most of the support software which is redeveloped does not differ functionally from what already exists.

The objective of SDSS is to address the control/proliferation/duplication problem and attempt to minimize those problems associated with the non-availability of militarized hardware during early software development design activities. The PDP-11/70 in SDSS addresses this problem in that the choice of this machine is based upon its architectural compatibility with the target system for the MCF validation effort. The approach to be taken for the validation effort is critiqued in Figure 2.

- DEVELOP CENTRALIZED SUPPORT SYSTEM FOR LIFE CYCLE SOFTWARE DEVELOPMENT AND MANAGEMENT OF TACTICAL COMPUTER SOFTWARE

- DEVELOP A SUPPORT SOFTWARE R&D FACILITY TO MINIMIZE DUPLICATION OF SUPPORT AND SYSTEM SOFTWARE

    - LANGUAGES
    - COMPILERS
    - OPERATING SYSTEMS
    - LINKAGE EDITORS
    - DEBUGGING TOOLS
    - DATA BASE MANAGEMENT SYSTEMS

- PROVIDE SINGLE UNIFORM INTERFACE TO (SDSS) USER/(MCF) DEVELOPERS FOR TARGET SYSTEM

- UTILIZE COMMERCIALLY AVAILABLE HARDWARE/SOFTWARE AS HOST ENVIRONMENT FOR TACTICAL SYSTEM SOFTWARE DEVELOPMENT

- UTILIZE EXISTING NETWORKING FACILITIES TO LINK DEVELOPERS TO HOST ENVIRONMENT

FIGURE 2. SDSS DESIGN APPROACH

As previously noted, SDSS will be established as a centrally located host facility at Fort Monmouth to control the development and test of the MCF System Software.

Figures 3, 4, and 5 depict the three distinct activity phases envisioned in its implementation.

PHASE 1

- ACQUIRE/INSTALL DEC HARDWARE
- INSTALL COMMUNICATIONS EQUIPMENT
- INSTALL/TAILOR PWB/UNIX
- TRAINING

FIGURE 3. SDSS - PHASE 1

PHASE 2

ESTABLISH OPERATIONS METHODOLOGY

- CONFIGURATION MANAGEMENT
- USER CONTROL
- TRAINING
- MALFUNCTION REPORTING
- MAINTENANCE
- DOCUMENTATION
- SECURITY
- ENHANCEMENT CONTROL

FIGURE 4. SDSS - PHASE 2

PHASE 3

- ARPA

- NEW TOOL INTEGRATION

- MILITARIZED PERIPHERALS

FIGURE 5. SDSS - PHASE 3

In Phase 1, the SDSS hardware/software and PWB/UNIX will be acquired, installed, and tested. In this phase, an initial plan of operations will be postulated to meet the specific requirements of the various user communities. The next phase (Phase 2) of activity will be concerned with the implementation of this plan with user control, configuration management, and a full operations methodology formalized. In Phase 3, enhancements to the basic system hardware and software will be made with particular efforts concentrated in the areas of networking, new tool development, and integration of specialized military hardware and peripherals into the system. These efforts will be driven by the main-line development endeavors of the MCF program.

CURRENT STATUS - CONFIGURATION

The Software Development/Support System (SDSS) to be implemented is based upon the projected programming requirements of three main user communities with functional responsibilities depicted in Figure 6. The computing facility established within the Center for Tactical Computer Systems (CENTACS) at Fort Monmouth will be made accessible to remote contractor programming stations located at the facilities of the developer of applications and training scenario software (User 1), at the facilities of the contractor responsible for the development of the DoD-1 compiler and language dependent tools (User 2), and at the facilities of the MCF hardware developer (User 3). This configuration is depicted in the block diagram of Figure 7 with communications between the CENTACS site and remote contractor locations through dialed telephone lines (dotted lines) and the ARPA-Network (solid lines). Figure 8 provides a detailed block diagram of this configuration.

USER COMMUNITY 1

APPLICATIONS SOFTWARE

TRAINING SOFTWARE

USER COMMUNITY 2

DOD-1 CONTRACTOR

- DOD-1 COMPILER
- LANGUAGE DEPENDENT TOOLS

USER COMMUNITY 3

MAINTENANCE & DIAGNOSTIC SOFTWARE

TRAINING SOFTWARE

FIGURE 6. SDSS - FUNCTIONS

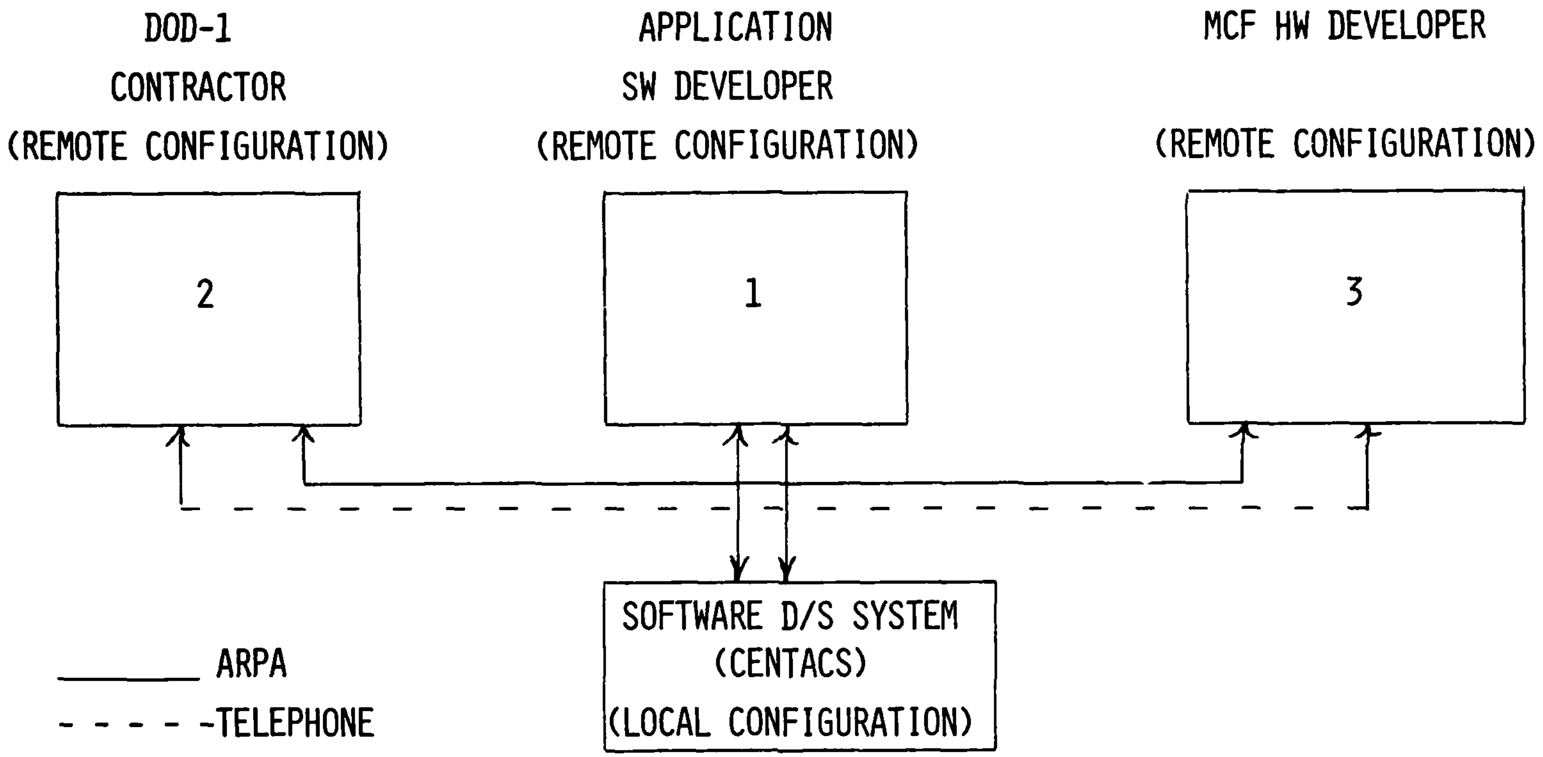


FIGURE 7. SOFTWARE D/S SYSTEM - MCF VALIDATION MODEL - NETWORK CONFIGURATION

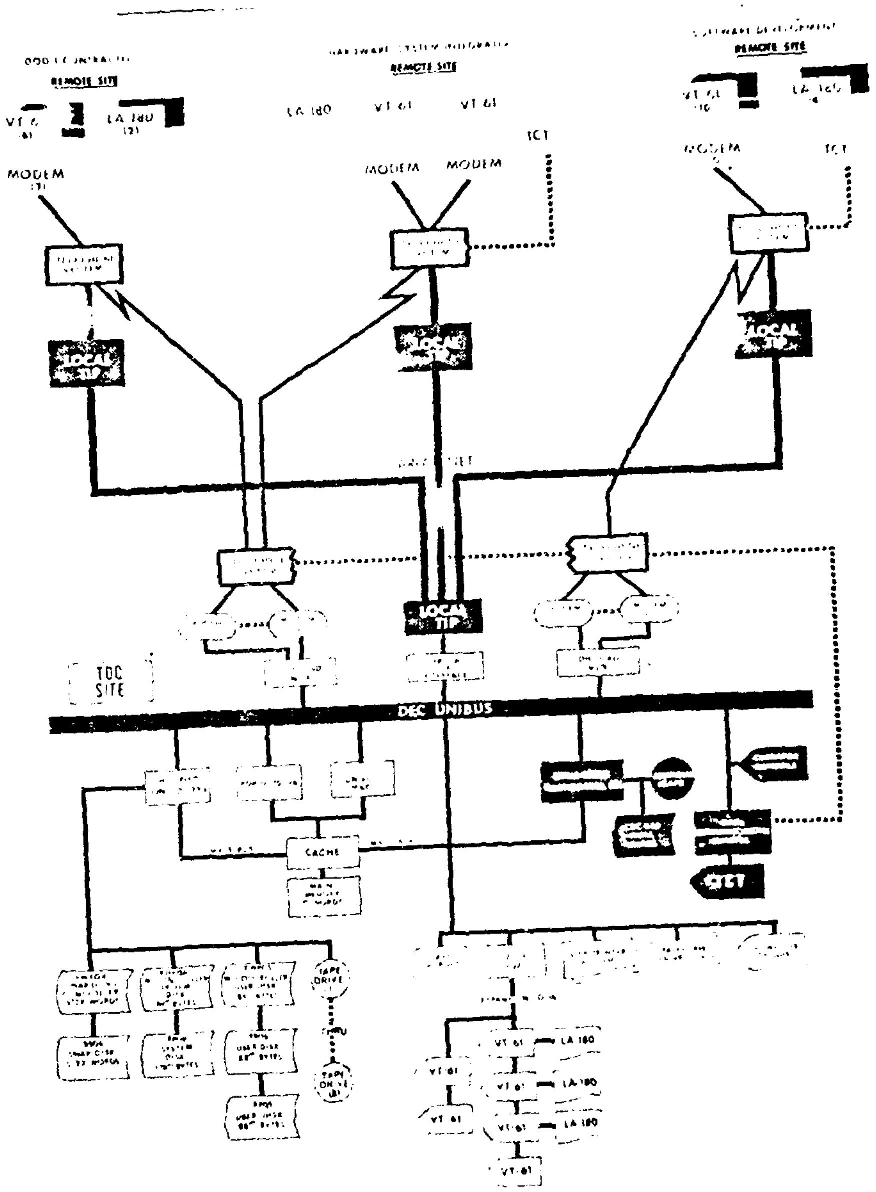


FIGURE 8. SDSS BLOCK DIAGRAM

As previously stated, SDSS will use the PWB/UNIX support software system to provide a convenient working environment and a uniform set of tools for the initial MCF computer program development effort. PWB/UNIX is a proven time sharing support system specifically engineered by the Bell Telephone Laboratory to provide an efficient, flexible, general-purpose multi-user program development environment. As depicted in Figure 9, it will provide SDSS with an effective file system, command language, source code control system, numerous languages and extensive text processing and documentation production facilities.

- FILE SYSTEM
  (MANAGEMENT OF USER/SYSTEM DATA, PROGRAMS, ETC.)
- PWB/UNIX COMMAND LANGUAGE
  (USER COMMUNICATION TO SYSTEM)
- DOCUMENT PREPARATION & TEXT PROCESSING
  (EDITING, FORMATTING, DOCUMENT PRODUCTION)
- SOURCE CODE CONTROL SYSTEM
  (PROVIDES AN AUDIT TRAIL OF ALL SOURCE CODE
  VERSIONS AND RELEASES)
- INGRES DATA BASE MANAGEMENT SYSTEM
- C COMPILERS & FORTRAN COMPILERS
  (INGRES & UNIX WRITTEN IN C)
- LANGUAGE DEPENDENT TOOLS
- YACC

FIGURE 9. PWB/UNIX MAIN COMPONENTS

Extensive augmentation of PWB/UNIX for SDSS is envisioned with the development of the DoD-1 compiler and a wide variety of specialized language dependent tools. The basic PWB/UNIX, available as an off-the-shelf software package for the PDP-11/70, is presently licensed to the Government on an unsupported basis under the provisions of a 19 Sep. 75 Software Agreement (with subsequent modifications) with Western Electric (Contract No. DAAB03-76-C-0182). Support, training, installation, and system enhancements will be obtained from external contractcrs.

Figure 10 provides a detailed delineation of the precise DEC hardware to be incorporated into SDSS.

QTY	DEC #	DESCRIPTION
1	11/70 - VA	Basic CPU, including 64KB, Decwriter
1	FP11 - C	Floating PT Processor
3	MJ 11 BG	250K Words - Memory
3	MJ 11 - BE	64K Words - Memory
1	RWP06 - AA	176M Byte Disk Pack Drive w/Controller
1	RP06 - AA	176M Byte Disk
1	RWP05 - AA	88M Byte Disk w/Controller
2	RP05 - AA	88M Byte Disk
1	RWS04 - KA	1M Byte Fixed HD Disk/w Controller
1	RS04 - AA	1M Byte Fixed HD Disk
1	TWE 16 - EA	Mag Tape, 800-1600 BPI w/Controller, 9Track
7	TE16 - EE	Mag Tape, 800-1600 BPI, 9 Track
2	LP11 - SA	800 LPM, 132 Col, 64 Char w/Controller
6	VT61 - AC	Video Terminal - 24 Lines 80 Char/Line, 128 Controller
1	DZ11 - E	Asynchronous Interface
1	H960 - DH	Cabinets, Cables, Chassis, etc.
1	DD11 - DK	Peripheral Mounting Panel
2	D11 11 - AD	Asynchronous 16 Line Max, w/Modern Controller
1	DR 11 - B	DMA Interface
1	PC 11	High Speed Paper Tape Reader
4	H312A	Modern Adapters w/EAI Interface
1	CR 11 EA	Card Reader

FIGURE 10. SDSS - HARDWARE CONFIGURATION

The facility will contain a PDP-11/70-VA with a memory capacity of 1M words of Core Storage, 618M bytes of Disk Storage, 6 Video Terminals, three with hard copy facilities, 8 Magnetic Tape Units, and various other card readers, line printers, and paper tape units. The two 176M byte disks are intended for general system use and will initially support the PWB/UNIX software development environment for the representative design. Contractors will each have a dedicated 88M byte disk in order to minimize the potential conflict in user requirements and to keep a clear separation between contractor individual job responsibilities. The number of video terminals provided at remote sites is based upon a 2:1 potential user to terminal ratio and will operate on communication lines with a baud rate of 300. The local configuration of 6 terminals has an expansion capability of up to 16 terminals. A tradeoff in the quantity of terminals supported versus response time will be carefully considered if additional terminals are added.

Figure 11 provides a delineation of equipments categorized as MCF devices that will be located at CENTACS to support on-line development of MCF software during the early representative design program development phase.

- 1 MILITARIZED STORAGE MODEL 640 DUAL DISK CONFIGURATION
- 1 RAYMOND MAGNETIC TAPE MEMORY WITH 4 CARTRIDGES
- 1 DATA COMMUNICATIONS CHASSIS (12 CHANNELS)
- 1 OPERATOR'S CONSOLE
- 3 TACTICAL COMPUTER TERMINALS (TCT)
  (ONE EACH AT AAI, CDC, & FT. MONMOUTH)

FIGURE 11. SDSS - TACTICAL DEVICES

The equipments include the Raymond Magnetic Tape Memory, the CDC 640 dual disk, the MCF Data Communication Chassis, and a local Singer Tactical Computer Terminal (TCT). Efforts will be required in the establishment of SDSS to interface these equipments to the DEC-UNIBUS. The objective is to provide a user interface for the remote site system developers that best simulates the MCF bus looking into the specialized devices and provides the capability of using the PDP-11/70 for actual program development. The Raymond magnetic tape unit will be used to facilitate the transfer of the operational software to the actual MCF equipments,while the CDC disk will be used if portions of such off-the-shelf software as DBMS or INGRES are adapted for use in the run-time environment. Similarly, the Data Communication Chassis and TCT devices will allow applications programmers to access actual MCF devices early in the development phase for scenario testing and debugging.

The communication plan is to have SDSS access the ARPANET through a Terminal Interface Processor (TIP), physically located at Fort Monmouth, for the majority of communications to remote user sites. The prime advantage of utilizing the ARPANET as the main communications media is that a distinct cost savings can be realized over use of dialed, WATS, or leased lines. In addition, the ARPANET will provide SDSS users with access to additional computer resources throughout the world.

ADVANTAGES

The SDSS approach will provide the Army and contractor/developers of MCF software with numerous advantages not fully realized by prior developers of tactical system software. In general, the primary facility requirements of tactical software developers (listed in Figure 12) so far as programming environment is concerned will be met. Namely:

(1) The continued availability of a full complement of computing services that are convenient, relatively inexpensive, and guaranteed during normal working hours.

(2) The availability of target system hardware early in the development phase.

(3) The availability of a facility that is human engineered with extensive, reliable, surprise-free tools.

(4) The availability of a facility with extensive document preparation capabilities.

(5) The availability of a facility that is adaptable to rapid organizational and personnel change.

(6) The availability of a facility which presents a stable user interface that masks detailed hardware considerations, including major configuration changes from them.

(7) The availability of a facility that utilizes a flexible, easy-to-use command language which is simple to learn.

(8) The availability of a facility that utilizes an efficient file system oriented to interactive use and guarantees controlled sharing of important system resources, including data and programs.

(9) The availability of configuration management tools that include a source code control system.

- AVAILABILITY
- HUMAN ENGINEERED, RELIABLE TOOLS
- EXTENSIVE DOCUMENT PREPARATION FACILITIES
- ROBUST TO ORGANIZATIONAL & PERSONNEL CHANGE
- STABLE USER INTERFACE
- FILE SYSTEM ORIENTED TO INTERACTIVE USE

FIGURE 12. SDSS - USER FACILITY REQUIREMENTS

SDSS will satisfy the above delineated general requirements, while at the same time addressing the problems previously presented. The specific advantages that will accrue to the Government as a result of the implementation of SDSS are outlined in Figure 13. In particular, advantages will be realized by the gain of effective control over the development software used for MCF while providing a stable, uniform, and efficient program development facility immune to development environment turbulence. SDSS will provide a repository for all development tools, operational software, test, maintenance, and training software packages used in the MCF development process. SDSS operations will be directly responsive to user/contractors, especially in cases where identified problems with support software obstruct the operational software development process. SDSS will impose severe restrictions on the enhancements of existing tools and on the introduction of new tools into the MCF software design environment. This will be done in order to best serve the overall MCF user community and to insure that all procedures and specialized software used by MCF contractor/developers are reproducible, Government owned, and readily available to all subsequent SDSS users and contractors. This is a major departure from past ways of doing business. Previous support software systems have rarely been portable across systems developments. Important support software components have often been seen to become proprietary items of their developers although developed initially for use on specific Government systems. This will be rectified by SDSS, although a considerable effort will be required to arrive at a uniform set of application independent software tools. In the past, the creation of such tools was made difficult by the differences in file structures, languages, communication protocols, and types and calibre of programmers that each contractor employs. SDSS will minimize all these variables by establishing the programming environment as a given long before the commencement of any MCF programming activity.

- GAIN BY EFFECTIVE CONTROL OF:
    - SUPPORT SOFTWARE & DEVELOPMENT TOOLS
    - OPERATIONAL SOFTWARE
    - TEST, MAINTENANCE, AND DIAGNOSTIC SOFTWARE
    - TRAINING SOFTWARE
- GAIN BY UNIFORMITY
    - SOFTWARE PROLIFERATION MINIMIZED
    - PROGRAMMER AVAILABILITY/MOBILITY ENHANCED
- GAIN BY INSENSITIVITY TO DEVELOPMENT ENVIRONMENT CHANGES
    - INSULATION FROM TARGET SYSTEM HARDWARE/SOFTWARE RECONFIGURATIONS
    - INSULATION FROM CONTRACTOR, GEOGRAPHIC CHANGES
- *GAIN BY EFFECTIVE SPECIALIZATION*
    - ESTABLISHMENT OF A BASELINE FROM COMMERCIALLY AVAILABLE COMPONENTS
    - CONVENIENT, AVAILABLE, ECONOMICAL
    - ABLE TO EASILY INCORPORATE STATE-OF-ART TOOLS, LANGUAGE
    - EFFECTIVE VEHICLE FOR EXPERIMENTATION

FIGURE 13. SDSS - ADVANTAGES TO GOVERNMENT

Through SDSS, effective gains will be realized by the minimization of duplicative support software. The transfer of Government and contractor personnel between various projects or between different phases of a particular project will most certainly be made easier by the uniformity imposed by SDSS. The effects of changes to hardware and software which often afflict both development and target environments simultaneously will be minimized with the SDSS approach. The majority of users will be fully insulated from perturbations in the host/target environment since the general purpose host facilities planned are completely compatible with the chosen architecture for MCF and are easily tailored to the specific target system configurations envisioned. In general, SDSS tools will not be tailored to any particular contractor's arbitrary preferences, although the unique function of each user most certainly will precipitate some very special tools. At the user interface the stability gained by establishing a baseline host environment from commercially available minicomputer software/hardware and by the methodical introduction of new specialized tools, languages, compilers, and operating systems will be particularly noticeable when new system functions, responsibilities, and tasks are assigned to SDSS. The existing user community will be minimally disturbed with these enhancements. Extensions to the basic system hardware are projected as being relatively economical investments in view of the relatively inexpensive cost of a PDP-11/70 processor compared to large or specialized host support systems. A situation where such a system extension might be required is in the area of system test. If the target hardware is unavailable at the time that load testing is to be performed and the existing facility overburdened, a relatively simple hardware extension to SDSS to generate the load and simultaneously run the application programs in a simulated target environment will be implemented. This extension will consist of the purchase of additional 11/70 processors and a physical reconfiguration of disk, core, etc.

CONCLUSION

Although the SDSS approach does not purport to be a panacea for all tactical software development problems, it does represent a bold attempt at bringing under Government control the root cause of much of what has been observed to have been wrong in such past activities, especially from the Government's point of view. The SDSS will provide the facilities whereby developers/contractors/users can get on a programming system quickly and build software quickly. It is intended that beyond the initial phase of providing a program development capability, SDSS will be enhanced to support such activities as configuration management, training, and post-deployment support for MCF. Much of the success of SDSS will depend on a user population willing to try new things and willing to provide the feedback necessary to adapt the baseline system into the facility which can be tailored to the Army's needs.

Each new feature to be added to SDSS will be consistent with existing features in order to maintain an environment that is simple, coherent, and conducive to productive use.

SOFTWARE ENGINEERING: TOOLS & METHODS II

Dr. Medhi Jazayeri
AIRMICS

SOFTWARE ENGINEERING

TOOLS AND METHODS II

SESSION CHAIRPERSON: Dr. M. Jazayeri

AIRMICS & THE UNIVERSITY OF NORTH CAROLINA

SESSION SUMMARY

This session reviewed some representative software engineering methods and their systematic application. Dr. Gerhart surveyed the current state of the art in program verification, the effect of program verification on software research and development, and some practical applications of program verification.

Dr. Rose talked about a microprocessor-based design tool which allows functional and performance simulation of a target system. Human factors issues are especially important in such a system and were emphasized in Dr. Rose's presentation.

Current Developments in Program Verification

Susan L. Gerhart

USC/Information Sciences Institute

After a decade of research, program verification is beginning to be applied to real, moderate-sized software components. Strictly speaking, program verification refers only to the activity of mathematically proving the consistency of programs with their specifications. However, the research has necessarily been broadened to encompass formal aspects of specification, program structuring, language definition, organization of programming knowledge, and programming methodology, all of which impact the verifiability of programs. Part of the talk emphasized this pervasive influence of program verification on software research and development. The talk also described current projects in evaluating and increasing the feasibility of practical applications of program verification.

CURRENT DEVELOPMENTS IN PROGRAM VERIFICATICN

Susan L. Gerhart
USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90291
213-822-1511

This work was supported by the Defense Advanced Research Projects Agency under Contract DAHC-15-72-C0308. The views expressed are those of the author.

Background

The activity many people call "program verification" got its modern start in 1967 with the paper [Floyd67], but the essential technical ideas and motivation had lain dormant for many years:

> "It is of course important that some efforts be made to verify the correctness of assertions that are made about a routine. There are essentially two types of method available, the theoretical and the experimental. In the extreme form of the theoretical method a watertight mathematical proof is provided for the assertion. In the extreme form of the experimental method the routine is tried out on the machine with a variety of initial conditions and is pronounced fit if the assertions hold in each case. Both methods have their weaknesses."
> ---Alan Turing, circa 1950, Programmers' Handbook for the Manchester computer

Until 1967, program verification, to the extent that it was discussed anywhere, was all experimental (testing). Since then, the theoretical method (proving) has developed significantly in theory, and, perhaps debatably, in practice. Unfortunately, the experimental method (testing) has remained comparatively stagnant and ad hoc, lacking either a strong underlying theory or much experimental justification. The social result is two opposing camps of researchers and practitioners, both grinding away at the weaknesses of their extreme methods, ignoring the benefits of the other method's strengths and failing to sufficiently exploit the strengths of their own method. While this paper will discuss primarily the theoretical method (proving), it is not meant to exclude totally the use of testing as a verification method. Indeed, the methods can and should be viewed as complementary.

The requirements for program verification by proving are rather simply stated: (1) the specifications for the program must be stated sufficiently precisely that correctness is determinable; (2) the semantics of the programming language must be sufficiently precise that a program's effects relative to specifications is determinable; and (3) there must be some overall proof theory which expresses how specifications and semantics together relate to some notions of correctness.

Much work has gone into attaining the "sufficiently precise" demands of (1) and (2). It was soon discovered that the semantics of many programming languages could not be made sufficiently precise - the languages were irrational, inconsistent, and horrendously complex. So the last five years has seen the development of languages which are more rational and less complex [LDRS77]. At least some newer languages (or parts of them) are defined sufficiently precisely that verification is not inhibited, but clean languages are no way near sufficient for verification. Also it remains to be seen whether the restrictions of these languages limit the scope of programming.

Likewise, much work has been done even more recently to develop languages (which, of course, must be precise) for stating user requirements and more detailed program specifications. The key here is abstraction - the omission of details - to allow specification to range through both external (user) and internal (system) views. Two techniques of interest are algebraic axioms (which will be discussed further later) and SPECIAL [Neumann77] based on what are called Parnas specifications. These emphasize change of state, values and constraints of input and output parameters, and inter-module information hiding.

Requirement (3) for verification has been well understood in principle for several years. The key idea is that of "inductive assertions" attached to loops to characterize the states of the variables, i.e. relations between them and properties of them, every time through the loop. Inductive assertions also appear as preconditions and as postconditions for procedures, characterizing the states at entry and exit of the procedure (or function). In this way, every program can be broken into straight-line parts between assertions. The semantics of the programming language both defines what these straight-line parts should be and how to formulate a theorem to the effect that executing the paths from one true assertion gives the truth of assertions at the other ends of the paths. These theorems are called "verification conditions". The major problem here, of course, is finding these assertions. There have been several variations on ways of stating assertions - intermittent assertions [Manna78] and subgoal assertions [Morris77], but a simple invariant assertion seems to work in most cases. Good surveys of the methods, tools, and difficulties of program verification are [London77,78] and [Luckham77].

Until very recently, program verification had been tried only on small examples, for very good reasons. First, the techniques were tentative and often didn't work very well. Small examples are adequate for rejecting various approaches. Second, reworking the same, small examples in various approaches provides a basis for comparison, again for the purpose of rejecting failures and confirming usefulness of not yet rejected methods. Third, the sheer volume of detail prevented scaling up very far in size and complexity of programs, even on the most promising methods. No one would read through this much detail and few researchers could invest the time to perform the experimentation and scaling up, especially when there were more possibilities to be explored on small examples. Fourth, the tools for program verification were inadequate for assisting on even small examples.

The point of justifying this earlier work is to make clear that this phase of research in program verification is nearly over. The current phase is actively exploring larger, more real, and indubitably relevant software components. The present phase still has to be viewed as experimental in character. We don't know when we try to verify a larger and more complex program whether we have either the intellectual or mechanical tools to complete the exercise. Indeed, the point of the experiment is to reject what doesn't work and improve what works partially. Furthermore, there are problems associated with large programs that aren't problems for small ones, e.g. managing a data base of theorems in various stages of proof and reducing the effects of changing one part of a large proof.

The rest of this paper will be a survey of current work on program verification at various research institutes. The survey is not meant to be comprehensive but only to convey the feeling of experimentation and scaling up just claimed. More detail will be given about one particular such experiment, the ISI Delta Experiment, only because it represents the author's efforts.

Microprocessor Research at IBM

[Carter78] argues that the enthusiastic rush to build more into hardware via microprocessors has lost sight of the essential quality of reliability. Hardware has traditionally been more reliable than software; now with built-in programs it can be equally unreliable, if the lessons of software unreliability are forgotten. One such lesson is that testing is inadequate for guaranteeing the absence of errors. The Microprogram Certification System (MCS) [Carter77] developed at IBM requires a complete formal description of the computer on which microcode is to run and provides a formal language with APL-like operators to do so. A microprogram is symbolically simulated in the MCS system to see whether the actual microcode (an array of bits) running (via the simulator) on the formally described computer produces the same results as a higher level description of the architectural specifications which the microcode is supposed to implement.

This system has been applied to several real microprograms, finding some very interesting errors. In one case, debugging code relating to the Test Support Equipment was found with the effect that system reset would not be performed correctly at certain moments. Other errors, or rather inconsistencies between specifications and code, were found in memory fetches in the last two bytes of memory and in a particular bit in an instruction. These errors are not always in the program but sometimes purely in the specifications, which of course can still cause trouble elsewhere.

Note: The thesis [Crocker78] developed a formalism and initial theory, called computational assertions, to represent microcode and specifications. It is being studied for application to code such as the IMP and the Fault-Tolerant Space Computer.

Hierarchical Design and Verification at SRI

The SRI Hierarchical Design Methodology (HDM) [Robinson77] separates design and verification into two stages: first that a formally specified design is consistent with a set of specifications and second that the implementation is consistent with the specifications. HDM has been applied to several operating systems: PSOS, the Provably Secure Operating System [Neumann77] ; SIFT, the software-implemented Fault-Tolerant System [Wensley76]; RTOS, the real time operating system; and KSOS, the kernelized Secure Operating System. Experience thus far demonstrates the feasibility of automatic verification of design properties, e.g. multi-level security or synchronization in SIFT. HDM is aimed at structuring both the system and the system development process, using formal methods to increase confidence in and control over design of large systems.

One SRI verification tool is RPE, the Rugged Programming Environment [Elspas77], which contains general translation techniques suitable for several languages, concentrating on JOVIAL. It is intended to become a tool for systems programmers.

The main verification tool is the *Boyer-Moore theorem prover* [Boyer77] which directly addresses a serious problem of theorem proving, extension away from built-in knowledge. For example, how much data structure knowledge should be build in if not all such knowledge can be? The fundamental logical system is recursive function theory.

This theorem prover has been used to prove: the Fundamental Theorem of Arithmetic; a tautology checker, i.e. another theorem prover; a simple arithmetic expression compiler and optimizer; the verification conditions for the Boyer-Moore fast string searching algorithm, on the average the fastest such algorithm known; and another parser.

Reliable and Secure Communications Systems at Texas

The Certifiable Minicomputer Project at the University of Texas [Good77] shows a blend of elements not present in the other projects: (1) a language, Gypsy [Ambler77], was designei especially for the purpose of specifying and verifying reliable and secure communication systems; (2) the supporting system includes tools for incremental development and verification of such systems; and (3) both run-time validation and proving are accepted for verification, omitting from verification run-time assumptions and using run-time assumptions to ease the complexity of proofs.

An example network specification and verification, incrementally performed, appears in [Moriconi78].

The Stanford Verifier

Almost all of PASCAL except floating point arithmetic has been implemented in the Stanford verifier [Luckham77], demonstrating that many language features can be formalized for verification. A special version of the verifier has been implemented for the detection of run-time errors in programs [German78]. This language has been extended to include modules and concurrent constructs.

The Stanford verifier's theorem prover contains a complete simplifier for quantifier-free formulas over the data structures common to programming languages. The Stanford research emphasizes the discovery and implementation of fast, specialized algorithms [Nelson77]. Other data structures and new specification concepts may be introduced by means of lemmas.

Numerous small examples have been verified and proofs of correctness of a parser and an operating system are in progress.

Security Kernel Verification at UCLA

The paper [Popek78] develops a model for data security in multiuser, shared resource computer systems, an application area where considerable expense is warranted to attain that goal. Current verification efforts include (1) a yet unmechanized, but formal proof of data security for several kernel calls, following an ALPHARD-like [Wulf76] methodology relating the abstract model to the implemented code and (2) proof that the PASCAL-like code implemented on the PDP/11 for each kernel call satisfies specifications given for it. This verification effort for about 1000 lines of PASCAL-like code is not yet complete; It brings in many aspects of operating system behavior that have not yet been formalized, e.g. the representation and mathematical properties of paging. There is no concurrency or parallelism in any kernel operation.

The ISI Delta Experiment

At the end of 1977, the ISI Program Verification project began an experiment to specify and verify a real-world software component. The first part of the experiment was to find a sufficiently large and real component which we felt had some chance of being verified. "Real-world" meant a running component that did something people really cared about. Side goals were chances of finding errors, intrinsic interest of the problem, overall benefits of formalization, and evolution of the ISI program verification system. Such a component was found in the ISI Military Message Experiment [ISI77] which is operational in Hawaii. The component is well-written in BLISS according to traditional programming style, but has no associated formalization, only prose comments beginning routines and an overall data structure diagram.

Modularization is fairly good, but the component does interact heavily with various other parts of the system, both more concretely in details of data representation and more abstractly in ways the component is used. Its size is around 1000 lines, with some use of macros, many system calls, few loops, and recursion.

A report on the nearly completed experiment is forthcoming [Gerhart78b]. Although not all the details of the component are handled (for reasons we will discuss later), the task and implementation have in no way been restricted to accommodate formal specification and verification. It was surprising that, while the task is common and the solution provided by the component is logical and feasible, we could find no written description in any computer science textbook or any articles in the technical literature.

The Delta Task

Briefly, the task is to permit several users simultaneously to modify files, called "Folders", which contain citations to messages and look roughly like trees. The system permits editing operations on folders expressed relative to these tree structures. In the message processing environment, a coordinator of a message may distribute the folder to other people for comment and editing. All people working on the same folder have individual copies of the folder during their editing sessions. At the end of each session, a list of net changes is generated and then later merged into the central copy of the folder by a "Coordination Daemon". This emphasis on small changes suggests the name "Delta". The delayed changes to be merged in look very much like the actual commands to change the file and are called "Delta Instructions".

The system restricts editing to two forms. At any time there can be only one user (the "modifier") making replacements, deletions, or internal additions to the folder, but there may be any number of users (the "appenders") making comments which are to be appended to the folder later. No further modifications can be made to a folder until the changes made in one modifying session have been merged into the central folder. The overall effect of this simultaneous modification must be that the list of changes produced at the end of a session accurately reflect the changes made during the session and that these changes are ultimately accurately reflected in the system copy. However, there is considerable latitude in the coodination. For example, someone might comment on a message in a folder, but by the time that comment is merged into the system copy a modifying user might have deleted the message.

The ISI Program Verification System

The ISI program verifier, AFFIRM [Musser79], accepts programs with PASCAL control and procedure structures (extended to include EUCLID-like [EUC] imports lists) and algebraically defined data types [Guttag77,78] as well as most of the regular concrete PASCAL data types. These abstract data types consist of functions with specified domains and ranges and axioms defining the effects of these functions, which either construct or modify an object of the type of interest or "access" a value outside the type of interest. The system's theorem prover is based on the use of these axioms as rewriting rules [Musser77]. Given a statement to be proved, the rules "reduce" or "simplify" the expression as far as possible by replacing left hand sides by right hand sides. This process requires that the axioms have an appropriate form to avoid loops in rewriting, produce the same results independent of order of application, and cover enough cases of reduction. We have various methods, both informal and implemented, to check and improve the rewriting behavior of a given set of axioms.

A mechanical proof looks very much like any mathematical proof. The user must state the theorem, find and state lemmas and indicate how and when they enter the proof, establish appropriate subgoals, reduce the complexity of intermediate steps by throwing away irrelevant information, etc. The user may also state induction schemas, e.g. structural induction, by which the system sets up the steps of an induction proof. Recording proof steps, suggesting next steps, undoing disastrous steps, redoing previous proof steps, etc. are all carried on by the system. Functions may be expressed recursively and then the definitions invoked explicitly during a proof. A great amount of planning must go into such a proof, since the system makes no effort to find proofs for the user (with the exception of algorithms for finding equality chains and instantiations of lemmas) beyond applying the rules of the axioms.

The earlier view of a mechanical PV system was more heavily oriented toward programs and verification condition generation, while AFFIRM treats the verification condition generator as a more subordinate component because there are numerous properties besides verification conditions to prove. The system is planned to evolve further to support a whole calculus of programs, e.g. with the ability to transform programs preserving correctness, and organized bodies of knowledge about programs, e.g. good axioms and lemmas about sets and sequences and pre-proved program schemas for common tasks. See [Gerhart78a] for discussion of these goals.

Verification Methodology

There is no technology for dealing with BLISS programs, because there is no formal definition of the language, which is heavily machine oriented. The Delta BLISS programs, however, were easily approximated by PASCAL+ programs acceptable by the system. The control structure of the BLISS programs (loops and procedure calls) is captured in the PASCAL+ programs, while abstract data types capture some of the data operations of the BLISS programs. For example, the writing of data onto files is mimicked by treating a file as a sequence, the blocks of data being read or written as indivisible parts of the Folder data structure. So at the BLISS to PASCAL+ level, we have only an informal argument that the two levels of program correspond, as given by a BLISS expert, and an ALPHARD-like [Wulf76] form structure treating the file as a data structure, with sequences of BLISS statements as implementations of operations of the abstract types.

At the next level up, the PASCAL+ programs look like any normal program, with the exception that the data types are axiomatically defined. Here the data types are sequences of DeltaInstructions split into opcode, data, and node identifier parts and trees modelling the folder and the operations performed on it during a user session. Assertions, preconditions, and postconditions are all predicates over the data types, usually condensed as notation recognized by the system as a separate pseudo-type. There is tremendous advantage to notating assertions, since assertions may be re-expressed for easier proving or to correct errors. In this case we chose to prove only that the PASCAL+ programs compute the same result, with their more file-oriented form of Delta Instructions, as some recursively defined functions, Merge and Changes, do with Delta Instructions.

This leads to yet another level where it is proved that the recursively defined functions interact in various ways. For example, there is a theorem to the effect that Merging the Changes made during a session into the file at the beginning of the session gives a file which looks like the one at the end of the session. This may seem like a simple theorem, but the algorithms for generating and merging Delta Instructions are actually quite complicated (although still briefly expressed recursively) because new identifiers must be created for nodes as they are added to the tree. Other theorems at this level express that this tree is well-defined, that empty trees lead to empty trees, and that appending interacts properly with modifying types of operations.

The highest level expresses (informally) the user's view of what happens to folders: that the modifications made during a session eventually appear in the central folder as they should given the other actions of the system. There is an informal proof that Merge and Changes do accomplish this user's view based on temporal assumptions about "fairness" and "lockout" scheduling. The overall structure of the proof is summarized in

level	language	data types	proved
requirements	English	Folder DeltaInstruction (undefined)	recursive functions satisfy user view
recursive functions	AFFIRM rules	Folder, Delta instruction	abstract programs implement functions (by assertion)
abstract programs	PASCAL+	Folder, linearized Delta instruction	concrete programs implement abstract programs (by forms)
concrete programs	BLISS	words, addresses	---

Results of the Experiment

Some of the results of this experiment are:

1. Requirements analysis. About half the effort went into finding out what the actual task was and how to express that formally. The top level mentioned above was nowhere described in the documentation given us. It was clear only that something was happening to a tree-like structure. This raises the practical question of what would happen during modification of the component by someone other than its author or others deeply familiar with the system. If the modifier produced a program sufficiently equivalent to the original there would be no problem, but modifying where equivalence is sufficient would be unusual. More likely, the modifier would need to preserve certain properties in order to maintain the integrity of the system. That is where this verification effort contributes the more formal specification which could help in maintenance of the system.

As mentioned earlier, we did a little literature survey and couldn't find any discussion of this problem, although it certainly must be familiar to many. Again, verification can make the contribution of forcing formalization in such a way that transferral of knowledge can now take place. People can study this particular solution and variations on it.

2. Specification formalization. We were able to get most of the concepts involved in the specification and implementation abstractly and formally expressed as algebraic axioms and recursive functions. There are about five pages of axioms, many of which follow the same form: definition for each constructor of the type. The user requirements and timing aspects of the system don't fit this specification method, but were easily handled in prose. The underlying storage model of pointers (especially new cells gained from free storage) was almost completely suppressed. One inherent problem of algebraic axioms, exacerbated by our system's orientation toward rewriting rules, is the expression of

legality of operations and errors. This simply requires a lot more study.

One aspect of the actual implementation is not mentioned anywhere in the specifications and is therefore ignored in the verification. Considerable pains are taken to check at the beginning of each BLISS routine that the arguments are in proper form, that folders are in buffers, that flags are set properly, etc. We have no way of specifying that such error-checking should take place or what to do in case errors are found. This is certainly one of the more open areas of specification (and of verification).

We have simplified the specification of the file updating task greatly in one respect over what the actual system does. In fact, the folder is a highly complex data structure with multiple linking. In the spirit of experimentation, we felt that the essence of the algorithm and data structure was sufficient for this experiment. Had we tried to specify every last detail of the code, we might have ended up trying to specify the entire SIGMA system (which is a worthy goal, but not within the scope of the experiment). We feel that what is not specified or verified could be obtained by extension from what we have now as axioms, abstract programs, and assertions.

This particular task of file updating has led us to discover a new form of description of an abstract data type: the constructor history representation. It is obvious that this is one way of defining many data types, but it has not occurred until now that this canonical form is of any use. Here we have used it to retain information for recursively defined Generate and Merge operations. We might have defined trees more as in [Guttag78] with additional assertions that there were series of commands that give this tree. Variations and trade-offs in definitional forms are another subject for future investigations.

3. Verification experience. The actual proof effort has been very tedious, but very beneficial, because the theorem prover at the start was new, with bugs, fewer user functions, and various space and time problems. This scaling up experiment has accelerated the development of the theorem prover and given us much valuable experience in using it which can be transferred next year to other users when the system is released.

4. Abstraction and structuring. The use of programs with abstract data types was the key to breaking down the complexity of the 1000 line program. These abstract programs are only about 200 lines long and immensely more readable than those in BLISS, especially with assertions and specifications. It is truly amazing that the essential algorithms can be expressed even more briefly in less than a page of recursive functions. What this demonstrates is exactly what has been claimed as the benefit of abstraction - omit enough details and you have a highly succinct expression of what is essentially happening. But succinct does not mean immediately understandable. Recursive functions may be very

hard to decipher, especially for persons not familiar with the style. However, these alternative presentations, especially as they are associated with levels of abstraction of data, give gains of understanding. Indeed, once we have these abstracted programs, we wonder how the program could ever be designed or understood without them.

What we hope to have accomplished is an existence proof: moderate-sized (1000 lines), significantly complex (hard to understand, weakly documented, and unpublished), real-world (written in older languages, interacting with both lower and higher system levels, not designed for verification) software can be formally specified and verified, not down to every last detail but revealing the abstract structure of the problem and implementation which either never emerged during or disappeared after design. Here is where our earlier observation about complementary verification methods comes to bear: this component has been thoroughly tested and used but until this experiment there was nothing beyond experience to convince us of its correctness. However, the verification could not capture all aspects of the problem, which are left to testing to convince us that the program is correct on omitted details and that the specifications are reasonable.

Summary

This survey of research in the development of verification systems and applications of verification to large-scale examples shows a diversity of methods being tried and problems being tackled. Microprograms, operating systems, communication systems, and data base systems are the primary software being investigated, both because these are the areas computer scientists like to study and because errors simply cannot be tolerated. Other important areas not being investigated on such a large scale are compilers and numerical programs, perhaps because the techniques aren't yet developed or the applications aren't as critical or testing and structuring suffice for the confidence level required.

Few of the systems have stabilized and many are rather specialized. The systems employ varying degrees of automatic proof finding and different styles of interaction. No clear best way for any special application has emerged, and it is likely that the next generation of systems will combine techniques experimented with in this generation.

Few of the larger applications of verification have been pushed all the way through. There is a tremendous amount of detailed work involved, especially when the supporting systems are still evolving. But most people working on these applications believe that their tasks can be completed, given the time and resources to do so. The major challenge is to reduce the complexity and difficulty of the tasks to practical levels, and that will take still a few more years.

However, one finds benefits accrue from simply attempting a verification, whether carried through to completion or not.

Formalization is immensely rewarding in the insights it brings and the complexity it reveals. The clean-up of languages has been prodded by the need to encourage verification and to remove the obvious barriers to it. The style of programming changes when one attempts verification. Good modularization is essential; indeed verifiability may be a measure of good modularization. One cannot escape the realization that there is so much more to be said about programs than what the actual text can convey. Good specifications and inductive assertions close the gap between what the program text says and what is needed to understand the program. The organization of knowledge about programs and the methods of teaching programming have been strongly influenced by verification. Indeed, we are finally seeing the emergence of a rational theory of programming, which can be used to better design and verify programs and systems.

Acknowledgements In response to a request for information, Peter Neumann, Bob Boyer, Derek Oppen, and Steve Crocker supplied descriptions of their work. Ralph London supplied useful criticism and claification. I am grateful to them for their help but they are not responsible for my interpretions and editting.

REFERENCES

[Ambler77] Ambler, A. L., Good, D. I., Browne, J. C., Burger, W. F., Cohen, R. M., Hoch, C. G., and Wells, R. E., "Gypsy: A language for specification and implementation of verifiable programs", in [LDRS77] , 1-10.

[Boyer77] Boyer, R.S. and J S. Moore, "A lemma driven automatic theorem prover for recursive function theory," Proceedings of the 5th International Joint Conf. on Artificial Intelligence, Boston Massachusetts, August 1977, 511-519.

[Carter77] Carter, W.C., Ellozy, H.A., Joyner, W.H., Jr., and Leeman, G.B. Jr. "Techniques for Microprogram Validation", IBM T.J.Watson Research Center Report RC6361.

[Carter78] Carter, W.C., Joyner, W. H., Jr., and Brand, D. "Microprogram verification considered necessary", National Computer Conf., 1978, 657-664.

[Crocker78] Crocker, S. "Computational Assertions", Ph. D. Dissertation, UCLA.

[Elspas77] Elspas, B., R. E. Shostak, and J. M. Spitzen, "A verification system for Jocit/J3 programs (Rugged programming environment - RPE/2)", Stanford Research Institute and Rome Air Development Center Technical Report RADC-TR-77-229, June 1977.

[EUC] Lampson, B. W., J. J. Horning, R. L. London, J. G. Mitchell, and G. J. Popek, "Report on the programming language Euclid," *SIGPLAN Notices*, 12, 2, February 1977.

[Floyd67] Floyd, R. W. "Assigning Meanings to Programs", In Symp. in Applied Mathematics, ed. J. T. Schwartz, 19-32, Volume 19, American Mathematics Society.

[Friertag77] R. J. Feiertag, K. N. Levitt, and L. Robinson. "Proving Multilevel Security of a System Design", *Proc. Sixth Symp. on Operating Systems Principles, ACM SIGOPS Review*

[Gerhart78a] "Program Verification in the 1980s: Problems, perspectives, and opportunities", ISI/RR-78-71, August 1978. Also *Oregon report on Computing: Problems of the 80s*, IEEE.

[Gerhart78b] Gerhart, S. L. and Wile, D. R., "Preliminary Report on the ISI Delta Experiment: Formal specification and verification of a file updating module", to appear.

[German78] German, S. "Automating proofs of the absence of common run-time errors", *Proc. 5th ACM Symp. on Principles of Programming Languages*, Tuscon Arizona, January 1978.

[Good77] Good, D. I. . "Constructing Verifiably Reliable and Secure Comunications Systems", Institute for Computing Science and Computer Applications Technical Report, Austin Texas.

[Guttag77] Guttag, J. V., "Abstract data types and the development of data structures", *CACM, 20*, no. 6, (June 1977), 396-401.

[Guttag78] Guttag, J. V., Horowitz, E., and Musser, D. R., "The Design of Data Type Specifications", *Current Trends in Programming Methodology*, Volume IV, editor R. Yeh, 1978, 60-79.

[ISI77] 1977 Annual Technical Report, ISI/SR-77-8, 7-24.

[LDRS77] *Proceedings of an ACM Conf. on Language Design for Reliable Software*, *SIGPLAN Notices*, 12, 3, March 1977.

[London77] London, R. L., "Perspectives on program verification," in *Current Trends in Programming Methodology*, Vol. II, *Program Validation*, R. T. Yeh (ed.), Prentice-Hall, 1977, 151-172. (Revised from "A view of program verification," *International Conf. on Reliable Software*, Los Angeles, California, April, 1975)

[London78] London, R. L. "Program Verification", In *Research Directions in Software Technology*, ed. Peter Wegner, (to appear) MIT Press.

[Luckham77] Luckham, D. C., "Program verification and verification oriented programming", in *Proceedings of IFIP Congress 77*, B. Gilchrist (ed.), 1977, 783-793.

[Manna78] Manna, Z. and R. J. Waldinger, "Is `sometime' sometimes better than `always'? Intermittent assertions in proving program correctness," Proceedings of the Second International Conf. on Software Engineering, October 1976, 32-39. Also Comm. ACM, 21, 2, February 1978.

[Moriconi77] Moriconi, M. S. "A system for incrementally designing and verifying programs", Information Sciences Institute Technical Report ISI/RR-77-65 and 66, November 1977. (Ph. D. Dissertation, Univerity of Texas at Austin).

[Morris77] Morris, J. H., Jr. and B. Wegbreit, "Subgoal induction," Comm. ACM, 20, 4, April 1977, 209-222.

[Musser77] Musser, D.R., "A data type verification system based on rewrite rules," Sixth Texas Conf. on Computing Systems, Austin Texas, November 1977.

[Musser79] Musser, D. R., The AFFIRM Program Verification Users Manual, to appear.

[Nelson78] Nelson, G. and D. C. Oppen, "A simplifier based on efficient decision algorithms," Proceedings of the 5th ACM Symp. on Principles of Programming Languages, Tuscon Arizona, January 1978.

[Neumann77] P. G. Neumann, R. S. Boyer, R. J. Feiertag, K. N. Levitt, and L. Robinson. "A Provably Secure Operating System: The System, Its Applications, and Proofs", SRI Final Report, Project 4332 (11 February 1977). 486 pages.

[Popek78] Popek, G.J. and Farber, D.A. . "A Model for Verification of Data Security in Operating Systems", CACM 21, no. 9, September 1978.

[Robinson77] L. Robinson, K. N. Levitt, P. G. Neumann, and A. K. Saxena, "A Formal Methodology for the Design of Operating System Software," in R. T. Yeh (ed.), Current Trends in Programming Methodology, vol. 1, 61-110, Prentice-Hall .

[Wensley76] J. H. Wensley, M. W. Green, K. N. Levitt, R. E. Shostak. " The Design, Analysis, and Verification of the SIFT Fault-Tolerant System", Second Int. Conf. on Software Engineering, San Francisco CA (13-15 October 1976).

[Wulf76] Wulf, W. A., R. L. London, and M. Shaw, "An introduction to the construction and verification of Alphard programs," IEEE Transactions on Software Engineering, SE-2, 4, December 1976, 253-265. See also Carnegie-Mellon University and University of Southern California Information Sciences Institute Technical Reports, 1976.

The Human Engineering of a System Design Environment for Microprocessor-Based System

Charles W. Rose
Donald C. Hewitt, Jr.

Case Western Reserve University

A design tool for the hardware and software of multiple microprocessor-based systems consists of a generalized meta assembler, a generalized linking loader, a register transfer language-based compiler for describing the system elements, an ecologist which links assembler and compiler outputs together with a topological description of the system into assimulation model, and a run time kernel which allows interactive functional and performance level simulation of the target system. The system is designed to run on a minicomputer with memory management under the UNIX operating system. User interactions which are controlled by the run time kernel fall into three categories: those which control the execution, those which define inputs and change the state of the simulated system, and those which request measurement of values or statistics and their subsequent display. The human engineering aspects of the kernel and the components of the system were discussed in detail.

THE HUMAN ENGINEERING OF A SYSTEM DESIGN ENVIRONMENT FOR MICROPROCESSOR-BASED SYSTEMS

Charles W. Rose & Donald C. Hewitt, Jr.
Case Western Western University
Cleveland, Ohio

ABSTRACT

A design tool for the hardware and software of multiple microprocessor-based systems has been developed which consists of a generalized meta assembler, a generalized linking loader, a register transfer language-based compiler for describing the system elements, an ecologist which links assembler and compiler outputs together with a topological description of the system into a simulation model, and a run time kernel which allows interactive functional and performance level simulation of the target system. The system is designed to run on a minicomputer with memory management under the UNIX operating system. Human engineering was a major design consideration.

INTRODUCTION

It is an axiom of software engineering that interactive software systems should be well-human engineered: that command languages and interfaces be clear, consistent, and forgiving; that on-line HELP and tutorial features be available that are accessible at the command level; that error messages and diagnostics be explicit, explaining the error and precisely identifying its location; and that the system work on behalf of the user, reducing where possible the volume of interaction, using context to simplify commands and saved prior state to prevent redundant typing and reprocessing.

When the interactive software system being considered is itself a tool for designing and developing computer hardware and software, human engineering issues become even more crucial. In addition to possessing the attributes listed above, a design tool 1) must preserve its users' freedom of expression and not significantly limit their creativity and 2) must create the impression that it is easier to design with the system than to design manually. A tool is useless if its potential users believe that the effort required to use it is not matched by the leverage it provides.

Such a design tool, N.mPc, is nearing completion at Case Western Reserve University. N.mPc is an adaptable software system to support the development of microprocessor-based systems [1]. The remainder of this paper discusses the motivation for the development of the tool, its structure and components, and describes in detail those human engineering aspects of the system as a whole and of its individual components which impact its effectiveness as a design tool.

BACKGROUND

During the last three years, interest in distributed intelligence computer systems has increased dramatically [2] [3] [4] [5] [6]. This interest has been generated by the availability of microprocessors with ever increasing performance/price ratios, and predictions of systems with still higher capability in the near future.

At the same time, however, the somewhat primitive state of development of a theory of distributed intelligence or multiple computer systems poses a problem. The control of these types of systems is not well understood, particularly for systems consisting of a large number of processors (more than 64). While there is considerable work in the area [3] [4] [6], it does not appear likely that the developing theory will significantly impact pratical system design in less than five years. Furthermore, ad hoc techniques for designing large multiple processors are not very advanced. To make matters worse, the design of microprocessor-based systems requires knowledge of both hardware and software technologies, few designers have both.

It is obvious, therefore, that design tools were needed which could be employed in the creation of these systems and which would support the skill level of the designer, provide insights into the structure and attributes of alternative structures, allow evaluation, and support developing theory. To meet these needs, a design and evaluation environment for microprocessor-based distributed systems should:

1. allow specification of multiple processor systems;
2. allow modeling at multiple levels;
3. allow changes to topology and microprocessor families with minimum work and expense;
4. possess monitoring and control facilities which can be employed anywhere in the target systems;
5. be useable by non-hardware or non-software specialists;
6. allow multiple processor families;
7. perform well when evaluating significant (in terms of the number of elements) target architectures.

None of the existing tools or methodologies including prototype implementation, conventional simulation, commercially available prototyping systems, or custom hardware prototyping environments possess all of the above attributes.

The need for a design facility embodying these seven characteristics was recognized at CWRU in 1975 and work on N.mPc was begun.

THE DESIGN ENVIRONMENT

N.mPc contains five separate tools which work together to produce a functional register transfer level simulation of multiple processor, heterogenous target systems. A system block diagram is shown in Figure 1.

A meta assembler, metaMicro [1] allows the user to specify the format, nmemonics, and associated bit patterns of the target instruction set. It maps nmemonics into bit strings, and outputs the instructions in a control/memory allocation graph which is machine independent.

A programmable, Generalized Linking Loader [7] resolves the machine dependent aspects of metaMicro graphs, links, and allocates the resulting image to physical memory according to user specified strategies.

The concept of machine independence is extended to simulation through a computer hardware description language (CHDL) compiler [8] which is used to translate processor and interconnection element descriptions into executable code.

This code, the output of metaMicro and the description of the topology of the target system are linked by an "Ecologist" [9] and "Simulated Memory Processor" [10] programmed into a simulation model which runs under the control of a Runtime Package [9]. The Runtime Package consists of a Command Interpreter, "Kernel" and "Simulated Memory Manager" (SMM). The kernel and command interpreter permit interactive control and monitoring of simulations, while the SMM manages the simulated memory contents, the available physical memory, and mass storage to optimize the performance of the simulation.

metaMicro

metaMicro is a general assembler for both vertically and horizontally organized target architectures. This generality is provided by the following facilities:

1) A Declaration Section - the structure and semantics of the target machine are described here.
2) An Instruction Section - the source program instructions to be assembled are described here.
3) An Include Facility - predefined source text may be included into an assembly to reduce the coding and input complexity. Declaration sections are often "included".
4) A Conditional Assembly Facility - this allows conditional assembly of instruction in a manner similar to that available in conventional assemblers.
5) An Extended Macro Facility - this allows an external/internal interface for ease of coding.
6) A Register Transfer Notation Option - this allows a source instruction format to use register transfer notation as well as the more common "Opcode-operand-operand" notation.
7) Illegal Opcode Checking - this allows the user to specify

illegal object instruction bit patterns which are then detected during the assembly process.

8) A Variable Length Capability - this allows the description of machines with variable length instructions.
9) A Global Label Facility - this provides the capability for linking object files.
10) Allocation Control - this provides information to the Linking Loader about requirements for absolute addresses and contiguous code sections.

metaMicro can assemble for machines having no more than a 256 bit-wide instruction and referring to no more than sixteen bits at any time. Its output is a "nodal graph", the structure of which conveys allocation information to the Linking Loader. While it is beyond the scope of this paper to introduce the detailed syntax and semantics of metaMicro, a well-documented example of the Declaration Section for the INTEL 8080 may be found in Appendix A.

Linking Loader

The Linking Loader/metaMicro subsystem is shown in Figure 2. The loader accepts as input nodal graph output files from metaMicro and a command program which defines the physical memory constraints and the semantics of the target machine. These semantics include an instruction and format declaration section identical to that required by metaMicro, the addressing modes of the target processor, and a set of rules which define how arbitrarily sized code segments in a vertical machine are to be relocated and how "next" addressing resolution is to be accomplished in a horizontal machine.

The command program is pre-processed by an Interpreter which adds information to allow tracing of the allocation process and creates internal commands which specify when logical to physical address translation is required. This allows relocation by the second component, the Allocator.

The Allocator is the process that combines metaMicro output files into an actual executable form according to the command program translated by the interpreter. The nodal file contains information specifying which instructions contain label references. The Transfer Declaration is used to specify the format of the unconditional transfers which may have to be inserted into the program in order to allocate a fragmented memory. The user may instruct the Loader to use a modified first fit, a fragmented memory, or low packing or high packing allocation strategy. After the allocation resolves all label addresses and allocates the available physical memory according to one of the four user-specified algorithms, it then binds the instruction label references according to the MODE declaration in the Interpreter program. The Interpreter programs for most microprocessors, vertical and horizontal, consist of less than two pages of code. The Interpreter program for the Intel 8080 is shown in Appendix B.

ISP' Compiler

In order to create the simulation model of a processor or other hardware module its structure and function must be specified in a form which can be compiled to executable code. For this purpose, a register transfer language, ISP', a modification and extension of ISP [11], and a compiler which converts ISP' module descriptions to executable PDP-11 code was developed.

ISP' was designed to allow the description of multiple processor/module architectures, a capability not found in other Computer Hardware Description Languages, (CHDL's). It contains the construct, PORT, which is an inlet (outlet) to (from) the environment external to the processor being described. Ports may be read and written by the processor in the same manner as any other storage elements. To allow easy synchronization between modules, a WAIT construct was added. WAIT suspends the execution of a simulation process until a specified condition occurs on a PORT. Also, a WHEN construct was included which allows the execution of an auxillary process WHEN a port condition holds. This is especially useful in the efficient simulation of interrupts.

Finally, events within a processor may be separated in time by the use of a NEXT TIME DELAY construct which specifies the execution or delay time associated with the previous register transfer statement.

An abbreviated ISP' description of the INTEL 8080 is shown in Appendix C.

Ecologist

The Ecologist, driven by a description of the target architecture (network) called the topology, generates an executable program for the simulation of the network. The topology of the network consists of a series of declarations that define the components of the network:

Signal declarations:
Define data buses used to connect the ports of hardware described in ISP'.

Processor declarations:
Create an instance of a processor or other hardware described by ISP', which can be referred to by name at runtime.

Time delay declarations:
Define the time unit to be used with the DELAY statement in a particular ISP' processor.

Connections declarations:
Connect the PORTs of an ISP' processor to signals defined in the signal declarations. By using field specifications, a port may be connected to a portion of a signal.

Initial declarations:
Bind a MEMORY of an ISP' processor to either a linking loader output file or to a UNIX file or device. The use of a linking loader output file is called a processed memory,

whereas a UNIX file or device is a raw memory. Raw memories allow the connection of an actual terminal or other peripheral to a simulation, to allow a natural user interaction with the simulated system.

After reading the topology of the network, the Ecologist finds and loads the ISP' output modules for each of the hardware components in the network, and forms simulated data buses to connect the ports of the hardware components. Finally, the simulation Kernel is loaded into the simulation.

In addition to the simulation program, the Ecologist creates two other output files. The runtime symbol table file contains all the information necessary to reference, at runtime, any structure defined in an ISP' source program or in the network's topology. The memory list file contains the names of all memory files required by the simulation, for later use by the simulated memory processor.

Simulated Memory Processor

Driven by the memory list output of the Ecologist, the Simulated Memory Processor (SMP) prepares memory files for use by a simulation. All linking loader output files needed by the simulation are converted from the packed format produced by the linking loader to the segmented or paged format required by the simulation program. Raw memories (Unix files or devices) found on the memory list file are not processed by SMP. In addition to the processed memory files, SMP creates a memory symbol table file containing the names of the memory files available to the simulation and the global labels defined in the memory files.

Figure 3 illustrates Ecologist and SMP subsystem in a block diagram form.

Runtime Package

The runtime package consists of the simulation kernel, the simulated memory manager, and the runtime command interpreter. Because of the limited (32K words) address space available to a program on the PDP-11, the runtime package has been split into three separate programs, each communicating through the UNIX interprocess communication mechanism, pipes.[1] The simulation kernel and ISP' output modules that make up a

[1] Under UNIX, cooperating processes may communicate using the pipe construct. A pipe resembles a file that is opened by one program for writing and by another program for reading. Data written to the pipe by the writing program is buffered and later read by the reading program. A read from a pipe with no buffered data causes the reading program to be blocked until data is written to the pipe. By using a pair of pipes, a bi-directional communication channel may be created between two programs. [12]

simulation reside in the same address space, communicating through pipes with the command interpreter and simulated manager, both in their own address spaces.

The simulation kernel performs the ISP' process scheduling and data manipulation functions required for the execution of the simulation. These functions are envoked by running ISP' processes through calls to several kernel routines. Because the kernel manages all scheduling and data manipulation, the user of the simulation gains access to the scheduling information and the contents of ISP' STATES, PORTS, and MEMORIES. In addition to the above mentioned tasks, the kernel also manages the breakpoints and monitor functions that allow the automatic collection of data from a simulation.

Simulated memories are handled effectively by the simulated memory manager. Through the use of one or two simulated memory managers communicating through pipes with the kernel, up to 128K bytes of simulated memory may be buffered in the main memory of the PDP-11 reducing, or in many cases, completely eliminating the need to access simulated memory on mass storage during a simulation. The simulated memory manager also handles raw memories, providing an optional prompting service that prints messages requesting or labeling data read or written by the simulation.

The interface between the user of a simulation and the simulation itself is handled by the runtime command interpreter. This program accepts commands from the user to examine or modify the simulation state, control the execution of the simulation, set execution breakpoints, and establish mechanisms that allow the automatic collection of data from a running simulation.

A block diagram of the runtime package, showing pipes as well as input and output files is given in Figure 4.

SYSTEM LEVEL HUMAN ENGINEERING

Of the components of N.mPc, only the runtime portion (Kernel, Ecologist, and Simulated Memory Processor) is interactive; metaMicro, the Linking Loader, and the ISP' compiler process files prepared in advance by the users, and their only direct interaction with the user is during invocation. In this section human engineering features common to these non-interactive components are examined. Properties specific to each will be discussed in later sections together with the human engineering features of the runtime environment.

Include Facility

Once a machine or module is selected as a component in a target architecture, it is reasonable for a user to expect to code the declaration section of metaMicro, the Interpreter Program of the Loader, and the machine description in ISP' only once, and to invoke them subsequently, whenever required, to assemble a source program, allocate and link a memory image, and perform a simulation. To accomplish this purpose, an Include Facility is included in the assembler, loader, and compiler which allows the temporary redirection of source input to the processor.

The format of the include statement is as follows:

```
include file_name$
```

include is the keyword that invokes the include mechanism. The file_name is a string that points to the proper file where the desired source text resides, and the statement is terminated by a '$'. This is the format of an include statement irrespective of its context. The processor will replace the include statement with the include file contents until end of file is reached, and then revert the source file back to the previous file. It is important to realize that all processors perform the include function exactly as stated above. The last character of the include file precedes the next character after the '$' in the include statement - no additional characters are imbedded. This can prove to be valuable, as in the case of including a general statement to which one desires to concatenate additional information. The concatenation will be performed correctly.

This facility is strictly checked for errors. A file_name must contain forty or less characters, exclusive of tabs, carriage returns, newline, and spaces. If the file cannot be found, a fatal error occurs and parsing is terminated. The facility also disallows the nesting of more than one level of includes, and this error is fatal to the assembly, loading or compiling process.

Based upon the implementation of the include facility, it is possible to place an include statement anywhere, with few exceptions, and have it recognized properly. The exceptions are: in a comment and in a string. All other occurrences of an include statement are allowed.

Using the Include Facility, libraries of element descriptions can be created and accessed on demand. Thus, the usefulness of the tool increases tremendously with use. When designing with elements already in the library, a user must only prepare metaMicro source code and invoke the assembler and loader; specify the topology and the element description files names to the Ecologist; and run the simulation.

UNIX - Compatibility

The invocation command formats are compatible with the general UNIX command structure so that any user familiar with UNIX will have no problem adapting to N.mPc. In addition, N.mPc follows UNIX file naming conventions: all metaMicro source files have a .M extension; all nodal output files have .N extensions; Interpreter source programs use .I; their interpreted forms use .A; etc. Using these extensions, the UNIX "wild card" file naming convention can be used so that, for instnace, the Allocater can be commanded to link and load *.N, or all files in the current directory with a .N extension.

Even with this capability, the invocation of the assembler and loader for several source files still requires considerable typing on the part of the user. The next feature further simplifies this process.

MAS-micro assembler

To further simplify the invocation process, a preprocessor, MAS was developed which allows the user to specify, in one single UNIX format command, the assembly and linking process for any one machine. MAS expects all .I and .A file libraries to exist in one directory so that, by specifying only the machine name, say I8080, MAS will select the I8080.A file if available or the I8080.I file if it is necessary for the Interpreter to process it. In addition, up to ten file names with .M and .N extensions may be specified together with execution options to the assembler and loader. MAS will cause metaMicro to assemble those files with .M extensions and then cause the link/loading of all .N files under control of the specified .A Interpreter program.

Using MAS, the amount of interaction with N.mPc is reduced to a base minimum. A similar facility, EC, has been developed to simplify the invocation of the Ecologist and Simulated Memory Processor. It will be described in some detail later.

COMPONENT LEVEL HUMAN ENGINEERING

metaMicro

As stated earlier and shown in Appendix A, metaMicro allows the developer of software for virtually any mini or microcomputer to define the structure and semantics of the target machine so that source programs can be prepared and assembled for the machine in either a conventional assembler format or in a register transfer format. The objective of this declaration section is to define the bit patterns which are logically "ORed" into an assembled instruction word when the associated mnemonics and operations are encountered in a source language statement.

In addition to the command language features, file extension conventions, and the include facility common to all N.mPc components, metaMicro possesses several other capabilities which significantly impact its usefulness as a software design tool. Several of these capabilities simplify the manner in which machine specification and source programs are described to the assembler, while other provide a rich feedback to the user. Among those in the first category are If's, Lists, Macros, BEGIN/END,SET/TES, and Illegal and Bind declarations. The Illegal and Bind declarations work together with a powerful Diagnostic facility to provide precise feedback on the nature and location of potential and/or actual user errors.

If Facility

The If Facility provides conditional control of the assembly process. It functions much like the Include Facility. The basic idea behind an IF is a conditional diversion of text, whereas an include is an unconditional diversion. The format of an IF statement is as follows:

if expression then {text} else {text}

expression is any valid metaMicro expression, text may be anyting, including another if statement. As each text operand must be delimited by '{}', there is no ambiguity introduced by nesting if's in then's and else's. The 'else {text}' portion is optional.

When the IF token is parsed, the expression is evaluated. If the expression is non-zero, the next source of characters will come from the text associated with the then '{}' pair whereas, if the expression is zero, the text between the '{}' pair associated with the else clause, if present, is the next source of characters.

Lists

A list structure is employed in metaMicro to define items with similar characteristics (i.e., global labels, macros, etc.). The structure consists of the definition descriptor (I.E., global, macro) followed by the items of the list separated by commas, with the entire list being terminated by a '$' (In general, the '$' character terminates all metaMicro statements.). Null lists are not permitted and a list may contain any number of items.

Lists are used in the 8080 description of Appendix A to specify formats, macros, illegal opcodes, etc.

Macros

To force bit patterns into fields within an instruction, there are two types of statements: field assignments and register transfers. To explicitly define those patterns each time their mnenomic or transfer operation is encountered would be extremelv cumbersome for reasonable applications. It was, therefore, necessary to create an interface between the form of mnemonics and operands that the user would like to use and the internal statements metaMicro requires. This interface is the macro processor, a very general, highly flexible facility.

The declaration keyword is macro and the declaration is in the metaMicro list format.

The format of a macro declaration element is as follows:

macro_name = macro_body &

macro_name is an identifier by which the macro will be referenced. Appended to the macro name may be a parenthisized parameter list with any number of parameters, each separated by ',' . Macro_body may consist of entire statements, other macro calls, partial statements, etc. In the 8080 example in Appendix A, macros are used to define the bit patterns to be logically Or'd into the instruction word when the various source op-codes and register specifications are encountered. In short, any information may be contained within macro_body including another macro. Macros may be nested to depths of 32. This statement can be made because a macro reacts much like the include facility. A macro merely indicates a temporary redirection of source input. That redirection is parameterizable for greater flexibility, but still amounts to a simple textual replacement. The macro facility metaMicro is an extremely powerful feature.

Begin/End

Since metaMicro and the Linking Loader are general tools, it must be possible to explictly specify those segments of code which must be logically and physically contiguous. The BEGIN/END construct allows the first specification.

For the purposes of metaMicro, there are two types of machines that may be defined: vertical and horizontal. A vertical machine is one that does not have a next address field associated with each and every instruction whereas a horizontal machine does have this field.

Assume that the field na is the next address field of each instruction.

```
instruction; na = . + 1$
instruction; na + . + 1$
.
.
.
instruction; na = start$
```

In this sequence, the user must supply the next address value for the collection of code. As a convenience, the Linking Loader will perform this function for the user as long as the instructions in question are located within a logical block (begin-end pair). In short, the user need only code the following to force the same flow of control as in the last example:

```
begin
     instruction$
     instruction$
     .
     .
     .
     instruction; na = start$
end
```

In the last instruction within this logical block of code, the user must provide his own next address field since the Linking Loader will not make an attempt to determine where to transfer control. It is entirely possible that a segment of code delineated by the begin-end pair may be the main instruction decoding loop, involving PC incrementation, instruction fetch, and the beginning of the decode phase. These instructions should be logically contiguous, but the Linking Loader cannot determine what to do at the end of the block.

Set/Tes

In microprogramming (horizontal machines), it is a common practice to dispatch to appropriate locations based on bit field patterns. This is a method by which hardware is designed, and design tools intended to facilitate programming for these machines must support this function. In essence, a user would like to guarantee that certain sections of code occupy successive memory locations, and this facility is present in metaMicro by using set-tes statements.

This dispatching causes a multiway branch to occur. After this demultiplexing operation, one normally wants to transfer, or multiplex, control to a common point. The ability to return control to that common point is another facility of set-tes statements.

Consider the following example:

```
          set
decode:   instruction$
          instruction$
          .
          .
          .
          instruction$
          tes
          instruction$
```

A user has decoded a field of an instruction and has decided to dispatch to 'decode' plus that field. The instructions contained within the set-tes pair will be allocated by the Linking Loader contiguously. In addition, the loader forces the next address fields in the instructions determined by the set and tes to point to the instruction immediately after the tes. If more than one instruction must be executed in the dispatch, BEGIN/END can be used to conveniently form compound instructions.

Illegal

Another function of assemblers is to disallow certain combinations of opcodes and operands. metaMicro also has this facility and the user has the ability to define which combinations are illegal via the Illegal Opcode Declaration. The user defines an instruction prototype of an illegal instruction and metaMicro checks each instruction formed in the instruction section with the illegal instructions defined.

It should be noted that the maximum instruction size of words is used each time that a check is made. If a match occurs, a warning message is generated. In all cases however, the instruction generated is always placed in the output file. As is the case with much of the declaration section, the illegal declaration is also in the form of a list, with the definition keyword being illegal.

The format of an illegal declaration list element is as follows:

(bit_pattern_formation) message

bit_pattern_formation must consist of field assignments and register transfers. Message is an optional string that is appended to a warning message which is generated when the instruction is encountered.

Bind

For some machines, many opcodes or instruction formats may be illegal for the same reason, and therefore, the same warning or error message should be generated for each. Were a string defined for each illegal opcode, extensive internal storage would be used, and the user would have been forced to do a lot of typing. To remedy this situation, the Bind Declaration was developed. Simply, a bind name is related to a string, allowing the string to be defined once and a name to be associated with it. Each time a specific string is required, the bind name may be used, at a substantial savings of storage. Again, a list structure is employed and the declaration keyword is bind. Bind is used extensively in Appendix A.

The format of a bind declaration list element is as follows:

string_name   message

The bind name is an identifier which may only be used in an illegal declaration. Message is a string and must not be null. However, the contents of the string may be null or of any length.

Error Messages

As metaMicro parses source text, errors may occur, prompting the display of error messages. There are three levels of error messages: warning, error and missing, and fatal. A fatal error terminates the parsing process whereas the others do not. In general, a warning message means that the results of the assembly process will probably execute correctly. An error or missing message indicates that execution is improbable, and a fatal error means execution is impossible. Error and fatal error messages should not be ignored as the linking process will not be attempted with these types of errors. However, a warning message does not affect the linking process.

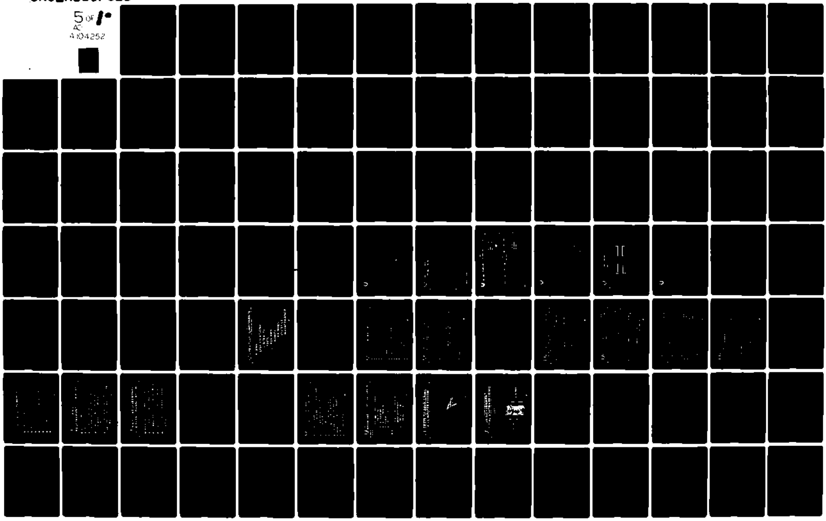
The general content of an error message is as follows:

degree module: message

AD-A104 252   INTERNATIONAL BUSINESS SERVICES INC WASHINGTON DC   F/G 9/2
US ARMY SOFTWARE SYMPOSIUM (2ND) HELD AT WILLIAMSBURG, VIRGINIA--ETC(U)
1978   S M TAYLOR   DAAK70-78-D-0030

UNCLASSIFIED   NL

5 OF /
AD
A104252



The degree portion is one of warning, error, missing, and fatal. Module denotes the area where the problem occurred, and is specific to the syntactic element level. The message portion describes the error, and is self-explanatory. If the listing option was disabled, messages also contain the source text line number causing the error.

metaMicro breaks the current source line at the point of trouble and then generates the appropriate message on the next line. The remaining text is placed on the next line after the error message, in such a manner as to line up correctly with its former location within the original source line. In cases where parsing may continue, subsequent errors within the same line are again broken in the same manner.

There is no limit on the number of errors detectable by an execution of metaMicro, but since metaMicro does know end of file, it will eventually terminate.

Linking Loader

The Linking Loader is unique. It allows its user to define in a concise manner not only the machine structure for which it is to allocate metaMicro nodal files, but also the allocation strategy to be employed. The fact that it obviates the necessity for machine dependent loaders to be designed for each new target machine is, in itself, a major human engineering accomplishment. In addition, the Linking Loader has other features which assist the user in specifying the operations to be performed, debugging the Interpreter command program and evaluating the output of the Allocator.

Space Override

The memory space declaration is part of the Interpreter program which is typically static and "included" when linking files for a particular machine. However, the allocation algorithm and available memory space may differ with target architectures. Therefore, the invocation command allows explicit specifications of the allocation algorithm, and also, an option to override the memory space declaration in the Interpreter program. If the override option is set, the loader will prompt the user at the terminal to enter a new space declaration for the purposes of the current linking operation.

Print Statement

The Interpreter command program language includes the capability of defining a Print statement at any point in the Interpreter program. A string is associated with the statement and is printed whenever it is encountered during execution of the program.

As instructions are allocated, the user may want notification of certain situations. The print statement may be used to display any information necessary to isolate the cause of a problem. Specifically, the print statement can indicate precisely which instruction is being resolved and any other information concerning the status of that instruction. Examples of its use are display of metaMicro program errors resulting in inconsistent Linking Loader input, allocation of instructions such that certain addressing modes cannot access those instructions, and trapping accesses into restricted areas of the target machine address space. Also, the print statement may be used to aid in debugging Linking Loader command programs.

Statement Trace

The ability to trace the flow of control during the execution of the command program has been designed and implemented into the Linking Loader. The Interpreter adds statement numbers directly into the interpreted command program file so that the allocator need only extract them when executing the command program.

Error Messages

An error message facility similar to that existing in metaMicro is incorporated into the Interpreter. It provides feedback to the user on syntax errors in the Interpreter command program.

Reporting Facilities

Since the logical address space created by any one metaMicro assembly is likely to be allocated non-contiguously by the loader because of memory constraints, and because this non-contiguous allocation may result in the addition of new instructions (transfers) or the modification of existing ones (changes next address field in horizontal machine instructions), the user requires certain information which will allow him to evaluate the effectiveness of the allocation and to debug the object program during execution. The Linking Loader provides several reports for this purpose.

The LOGICAL SPACE MAP shows how the logical spaces are allocated in the target machine address space. The entries in this and other reports are generated by showing each and every subspace division determined by both mapping and allocation. The LOGICAL SPACE MAP report lists divisions in the order in which the logical spaces are presented to the allocator whereas the PHYSICAL SPACE MAP report lists divisions in increasing order by absolute address. With these two reports, a user can easily determine where code is located by file name as well as what code is located at a certain address.

The EXTERNAL SYMBOL TABLE report shows where global labels are located in the machine address space and the logical spaces that have referred to them. Unreferenced global labels indicate their unreferenced status. The FREE SPACE AFTER ALLOCATION report shows where areas in the target machine address space have not been used to allocate the current mix of logical spaces. The free space indicated plus the space allocated to logical spaces plus the space used by unconditional transfers in the vertical case will always equal the space defined in the space declaration of the Linking Loader command program. The TRANSFER LOCATIONS report lists the new transfer instructions which were added by the Allocator. It indicates the physical location of the instruction and its logical target.

These reports together with the Trace and Print facilities, provide the user complete feedback on the Linking Loader process and its results.

ISP' Compiler

The ISP' Compiler is patterned after metaMicro from a human engineering standpoint. It, too, possesses a precise Error/Warning Message facility and supports the Include function.

In addition, it also provides several constructs which make possible the straightforward description of the synchronizaiton of concurrent independent processors or elements during a simulation. These constructs were mentioned briefly in an earlier section. A short example of their use will be presented here to demonstrate their power.

```
%
 WRITER
%
port resume,bus,ready;             ! Three ports declared
state R;                           ! A register,R
main := (                          ! Begin main process
        bus = R; next(deskew)      ! Put data, delay for settling
        ready = 1;                 ! Send ready signal
        wait(resume:lead)          ! Wait for resume to rise from
                                     READER
        ready = 0; next            ! Respond to resume from READER
        wait(resume:trail)         ! Complete handshake.  Resume falls.
        main                       ! End main process
        )

%
 READER
%
port resume,bus,ready;             ! Same three ports. Assume they are
state P;                           ! Connected.
main := (                          ! A register,P
        wait(ready:lead)           ! Wait for data present. Ready rises.
        P = bus; next              ! Read data when ready
        resume = 1; next           ! Signal data consumed
        wait(ready:trail)          ! Wait for WRITER response. Ready falls.
        resume = 0; next           ! Complete handshake
        main
        )
```

This writer/reader example typifies the transfer of data on an asynchronous bus. The writer places a data word onto the bus, then waits for a data deskewing period before signaling "data ready". The writer process suspends operation until the resume flag is asserted. The reader, which has been waiting on the "data ready" flag, begins operation and reads the data from the shared bus. The reader asserts the resume flag acknowledging the reception of the data, and waits for response from the writer process. Upon sensing the leading edge of the resume signal, the writer drops the ready signal and awaits the lowering of the resume flag. The interlock is completed as the reader awakens and clears the resume flag, thus reawakening the writer, which repeats the writing operation.

System timing specifications have been added to ISP' by extending the "next" separator. The reserved word " delay" may be followed by a time constant which represents the execution or delay time associated with the preceding register transfer. The ISP' statement,

```
R = P; delay(400)
```

has the interpretation: "Transfer the contents of register P to register R. Then suspend process execution until 400 time units have elapsed."

Ecologist

The strong point of the Ecologist is its simplicity. To construct a simulation program for a network, the user of the system simply creates a file containing the topology of the network, generates the necessary ISP' and Linking Loader outputs, and runs the Ecologist. Appendix D illustrates the topology of a simple 8080 system, based on the 8080 described in Appendix A, connected to simple memory, clock, and I/O modules.

To allow the use of libraries of standard ISP' descriptions, the Ecologist is capable of generating full path names (names of UNIX files, including specification of directories and subdirectories) from simple filenames. A library directory may be specified when the Ecologist is invoked, so that when the Ecologist cannot find an ISP' output file in the current directory, it searches the library directory for the file.

Simulated Memory Processor

Driven by the memory list output of the Ecologist, the simulated memory processor requires very little input from the user. To run the simulated memory processor, the user simply specifies the name of the simulation for which memories are to be processed and an option letter specifying the page size to be used for the processed memories. A listing option is available that cause SMP to print out a verbose listing of its actions.

EC, A Utility For The Execution Of The Ecologist And SMP

Because both the Ecologist and SMP must be run to create a simulation, a utility, EC, has been developed that executes both of these programs. EC is invoked by specifying a simulation name to be passed to both the Ecologist and SMP, options to both the Ecologist and SMP, and an optional directory name that is passed to the Ecologist for use as the ISP' library directory.

To allow the use of EC to run the Ecologist or SMP individually as well as together, EC asks its user whether the hardware of the simulation, the software of the simulation, or both are to be changed. If hardwar is specified, the Ecologist is run, and if software is specified, only SMP is run.

The Runtime Package

The runtime command interpreter implements the interface between a running simulation and the user of the simulation. User commands are interpreted and transmitted to the kernel in an encoded form, thus removing the space consuming interpretation process from the address space of the simulation, as well as allowing room for the development of a more powerful command set. Figure 4 illustrated that the command interpreter has access to the runtime symbol table file and the memory symbol table file, permitting user commands to specify structures in the simulation by name, rather than by machine generated number.

The interface to the simulation is easy to use, consistent, and provides the user with access to all aspects of the simulation. The command set provided to the user is complete and concise.

It has a number of features developed for user convenience. A "help" command is available that can either list the available commands or give detailed instructions for the use of a specific command. Commands, in general, consist of a keyword used to specify the command followed by a list of parameters to the command. Use of the command set if simplified by allowing the abbreviation of commands to the shortest character string that uniquely defines the command. Other conveniences include: the ability to input values in ascii, binary, octal, decimal, or hexidecimal; the ability to specify output in any of the bases; and addressing of memory by either numerical subscripts, labels, or both.

Files of command interpreter commands may be executed through the "include" command, which allows many levels of nesting. Included files may be used either for the initialization of structures in a simulation prior to execution, for the initializaiton of frequently used breakpoints or monitors, or for the examination of multiple structures at runtime.

There are five basic types of interaction between the user and the simulation.

Simulation Control

The simulation, when started, locates its various symbol and memory files without user intervention. At any time, the user may save the complete state of the simulation, and either continue execution of the simulation or terminate the simulation. The simulation is restartable from any saved state.

The user starts the simulation by running the program produced by the Ecologist.

The simulation is stopped with the "quit" command.

The "save" command causes the current state of the simulation to be written to a file and made executable, so that the simulation may be restarted from that state at a later time.

Through the "msave" command, memory files may be saved individually or collectively.

State Manipulation

Any structure defined in ISP' or in the simulation topology is available for examination or modification by the user. Structures are referenced by name, and memories allow indexing on labels defined in the metaMicro programs they contain.

Any ISP' structure (state, memory location, or port) may be examined through either the "ascii" or the "examine" command. The structure is specified by giving the processor name, defined in the topology of the network and the ISP' structure name. For example, to examine the state, "pc", in the processor, "cpu", the following is typed:

```
examine cpu : pc
```

In addition, a default processor may be defined through the"setproc" command, as illustrated in the following set of commands that examines the structures, "pc", "ir", "address", all in the processor, "cpu":

```
  setproc cpu
  examine : pc
  examine : ir
examine : address
```

Any ISP' structure may be modified through the "deposit" command. To deposit a value of 32 in the pc of cpu:

```
deposit 32 cpu : pc
```

Ascii strings may be used as values in the deposit command, as illustrated here;

```
deposit "a" cpu : a
```

Signals may be examined by name through the "signal" command.

Memory locations may be examined with the help of labels defined in the metaMicro programs stored in the memory. Three address formats are allowed, labels, numbers, or both.

```
examine cpu : mainmem[ start ]
examine cpu : mainmem[ 4 ]
examine cpu : mainmem[ start - 3 ]
```

Execution Control

The "status" command prints, for each ISP' process, the name of the processor to which the process belongs, the source listing line number where the process is currently executing, and the state of the process. A process may be either running, ready to run at a future time, or waiting for a port condition. The port name and condition are printed for processes waiting on a port, and the activation time is printed for processes waiting on time. The "status" command may also be used to examine the statue of either running, ready to run, or waiting processes only.

Execution breakpoints, which stop the simulation when the specified condition holds, may be set either on time values or on structure conditions through the "bkpt" command. To set a breakpoint on time, the user specifies the amount of simulation time, in nanoseconds, before the activation of the breakpoint, as in the following example,

```
bkpt 1000
```

which sets an execution breakpoint 1000 nanoseconds from the current simulation time. To set a breakpoint on a structure, it is necessary to specify the structure, as in the "examine" or "deposit" commands, and a breakpoint condition. The available breakpoint conditions are: any read from the structure, any write to the structure, a change in the structure value, an increase or decrease in the structure value, as well as a write to the structure that causes the structure to be equal to, not equal to, greater than, less than, greater than or equal to, or less than or equal to a specified value. Any combination of these conditions may be used, as long as only one of equal, not equal, greater than, etc. is used. For example, to generate an execution breakpoint when the program counter of processor ioproc either decreases or becomes greater than 400:

```
bkpt ioproc : pc decrease gtr 400
```

In addition to the breakpoint constructs listed above, breakpoints may be repeated automatically by using the "repeat" prefix to the bkpt command. Breakpoints on time, when repeated, generate a series of breakpoints separated by a fixed amount of time. Breakpoints on structure values, which normally are removed after one activation, are not removed when repeat is specified. An example of the repeat command, designed to stop the simulation each time the 8080 input instruction is executed (op code 11011011) is:

```
repeat bkpt : ir eql 0b11011011
```

Note the use of a default processor assignment through the omission of the processor name in the specification of ":ir". To allow more complex breakpoint mechanisms, breakpoints may be serially linked using the "after" specification. The keyword, "after", establishes a list of breakpoints, which affect execution as though the last breakpoint on the list were established, the simulation run until the breakpoint causes execution to stop, the next breakpoint then established, etc. For example, the command,

```
bkpt 1500 after cpu : pc eql 40 after : intr change
```

used to stop execution 1.5 milliseconds after the pc of the processor cpu reaches 40, after the intr flag changes, is equivalent to the series of commands,

```
bkpt : intr change
run
<wait for execution to stop>
bk cpu : pc eql 40
run
<wait for execution to stop>
bk 1500
```

The "after" specification may be used with repeat to cause a complex repeating breakpoint.

All bkpt commands return a number that is the number of the breakpoint. When the breakpoint causes the simulation to stop, the message,

```
Simulation halted by breakpoint <number>
```

is printed.

In order to find what structure or time value caused the activation of a breakpoint, the whatis command is used. This command echos the command typed by the user while setting up the breakpoint under question. For example, to determine what caused the stop of execution above, the user may type:

```
whatis 39
```

The whatis command may be used at any time, not only immediately after the breakpoint stops execution.

The "remove" command is used to remove breakpoints that have not yet been activated or to remove repeating breakpoints. The remove command requires as a parameter the number of the breakpoint to be removed, as in the following command to remove breakpoint 39:

```
remove 39
```

The "run" command simply begins or continues the execution of the simulation. The execution continues until an execution breakpoint is activated, or until the keyboard interrupt key is pressed by the user of the system.

Automatic Collection Of Statistics

Monitor facilities are available to automatically collect and display or store data from a running simulation. The basic commands available are" "display", "trace", "timer", "sum", and "maxmin".

The display command is used to display the contents of a structure on a terminal during a simulation. A display is initiated by typing "display" and a structure name, as shown here:

```
display cpu : ir
```

The command interpreter returns a display number which is used to identify the display. Each time a running simulation writes to the structure being displayed, a line is written to the terminal containing the display number, current simulation time, and new structure value. Multiple displays may be active at any time.

The trace command is similar to the display command, but writes a record to a trace output file instead of the terminal. This file may be post processed to yield trace listings, timing diagrams, histograms, etc.

Both traces and displays may be enabled and disabled by breakpoints, and the "repeat" prefix may be used to "repeat" the enabling and disabling process. The following example illustrates these functions:

```
repeat display cpu : a enable cpu: pc gtr 5 disable 400
```

This display causes the structure "a" of cpu to be displayed for 400 nanoseconds after each time the pc of cpu receives a value greater than 5. If "repeat" were not specified, the display would be enabled once, and then removed 400 nanoseconds later.

By using the "sample" option, a trace or display may occur on a condition other than a write to the structure under observation, for example,

```
display cpu : pc sample cpu : ir write
```

causes the pc of cpu to be displayed each time the ir of the same processor is written. Repeat, enable, and disable also function with sample.

Timers are used to obtain the time difference between two events. A timer is specified by a start and stop breakpoint, as well as an optional trace or display directive. Timers normally display the simulation time difference between the start and stop breakpoints, but they may be directed to write this data to the trace file. The following command is used to find the time required to execute the instructions between location 40 and location 50 of a program:

```
timer start cpu : pc eql 40 stop cpu : pc gtr 50
```

The summation and maxmin commands examine a structure each time it is written, and either collect the sum of all values or find the maximum and minimum values of the structure. The sum or maximum and minimum may be displayed by executing the summation or maxmin commands and specifying the sum or maxmin number that is desired. For example, the following series of commands is used to find the maximum and minimum values of a stack length register during a 2 millisecond period:

```
maxmin : stklen
<number 34>
bkpt 2000
run
<wait for execution to stop>
maxmin 34
```

Any of the above data collection mechanisms may be removed, along with their associated breakpoints, by executing a remove command and specifying the number of the mechanism to be removed. In addition, the whatis command may be used to find the command used to initiate any data collection mechanism.

Error Reporting And Recovery

The system recognizes and gracefully recovers from any runtime error.

Any runtime error, say, an illegal subscript or deadlock condition, results in an error message printed to the user. All related ISP' structures are explicitly named, and the current line number in the ISP' description of the simulated processor is printed. No recovery action is taken other than returning control to the user to allow either further exploration of the fault or a restart of the system.

Errors in commands typed by a user result in descriptive error messages from the command interpreter. Illegal or erroneous commands do not modify the simulation state in any manner, so that the careless user cannot easily destroy a simulation.

Through the use of raw memories, the simulated memory manager may also communicate with the user. If a simulation contains raw memories (ISP' memories connected to UNIX files or devices), the user is prompted at the beginning of the simulation to determine whether the assignment of raw memories to UNIX files and devices is to be modified. This prompting allows the user to connect a simulation to a different terminal than originally planned without rebuilding the simulation through the Ecologist and SMP. In addition, raw memories may be instructed to prompt the user before reads or writes thus allowing the routing of multiple ISP' memories to a single terminal without confusion.

The command set described above provides a valuable user interface to a simulation. Complex activities, such as the production of timing diagrams, or the production of "Logic Analyzer" type output, are not performed at runtime, to save on the time and space requirements of a simulation. Post processing of trace file data provides an open ended mechanism for the extraction of data from a simulation, without requiring additions to the runtime package itself.

SUMMARY AND CONCLUSIONS

N.mPc has been created as a design tool for multiple processor hardware/software systems. Its intent is to allow system designers to evaluate alternative architectures in a reasonable time and without committing to expensive special-purpose development hardware.

The human engineering of this tool was a primary design consideration. Care has been taken to minimize the amount of unessential user interaction while at the same time, not sacrificing user freedom of expression and creativity. Furthermore, while concise Error and Warning facilities are incorporated, the user is allowed to specify additional feedback if he desires.

In March, 1978, Case Western Reserve University was awarded a contract to complete the implementation of N.mPc on a PDP-11/70 under UNIX.

metaMicro and the ISP' compiler, together with the Linking Loader have been completed. metaMicro and the Loader have been delivered.

The Ecologist is working and most of the kernel is also. Simulations have been running since August, 1978. Implementation should be complete in November, 1978. Preliminary results indicate that N.mPc can simulate about 200 8080-class machine instructions (about 4μ sec) per elapsed time second on a PDP-11/34 implementation of N.mPc. This will translate to about 1000 instructions/second on the target PDP-11/70, or a simulation to real time ratio of about 250:1.

This number represents the total simulation rate for N.mPc. Therefore, as the number of processors increases, say to 4, the simulation time required to execute 1000 instructions/processor rises to 4 seconds. Similarly, if the real execution speed of a target processor increases, say to 1μ sec average instruction execution time, the simulation to real time ratio rises to 1000:1.

Even so, the performance is acceptable when one considers that other instruction level simulator packages [13] [14] and [15] have ratios on the order of 1000-5000:1.

This quite good performance can be achieved for all but extremely large target architectures. By using UNIX "pipes" to communicate between simulated memory and processor instances, the 32K word process address space restriction imposed by the PDP-11 architecture can be subverted without resorting to storing simulated memory pages on disk in most cases.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the contributions of P.J. Drongowski, who developed many of the concepts of N.mPc in his masters thesis; L.R. Rogers, who conceived and implemented the Linking Loader; R.V. Straubs, who implemented the ISP' Compiler; G.M. Ordy, who developed SMP and SMM; and F.I. Parke, who has been the Principal Investigator on the implementation phase of N.mPc; special thanks to Ms. C. LesCook, who typed the manuscript.

REFERENCES

1. P.J. Drongowski, "N.mPc: An Adaptable Software System to Support the Development of Microprocessor-Based Systems", Andrew R. Jennings Computer Center Report #1177, Case Western Reserve University, 1977.

2. C.W. Rose, and J.D. Schoeffler, Distributed Intelligence and Input/ Output in Data Acquisition Systems, Proc. International Telemetering Conference, Vol. XII, 705-720, September 1976.

3. R. Cooper, "The Distributed Pipeline", IEEE Transactions Comput., November 1977.

4. Proceedings of the 1976 International Conference on Parallel Processing, IEEE, 1970.

5. A. Avizienis, "Architecture of Fault Tolerant Computing Systems", Proc. Intl. Symposium on Fault Tolerant Computing, 1975.

6. S.S. Reddie and E.A. Feustal, "A Conceptual Framework for Computer Architecture", Comp. Survey, Juen 1976.

7. L.R. Rogers, "A Generalized Linking/Loader for the Allocation of Code in Vertical and Horizontal Machines:, Case Western Reserve University, 1978.

8. R.V. Straubs, "A Compiler for a Register Transfer Based Simulation Language", Case Western Reserve University, 1978.

9. D.C. Hewitt, Forthcoming Masters Thesis, Case Western Reserve University, 1978.

10. G.M. Ordy, Forthcoming Masters Thesis, Case Western Reserve University, 1978.

11. G. Bell and A. Newell, "The PMS and ISP Descriptive System for Computer Systems". Proc. AFIPS, STCC, 1970.

12. D.M. Ritchie and K. Thompson, "The UNIX Time Sharing System", Communications ACM, Vol. 17, No. 7, 365-375, July 1974.

13. B.O. Aygen, "Dynamic Analysis of Execution: Possibilities, Techniques, and Problems", Ph.D. Dissertation, Carnegie-Mellon University, September 1973.

14. C.W. Flink, "A Microprogrammed Environment for a Software Development System", IEEE Compcon 75, Fall Digest 1975.

15. C. Vickery, "Software Aids for Microprograms Development", Micro Proceedings, September 1974.

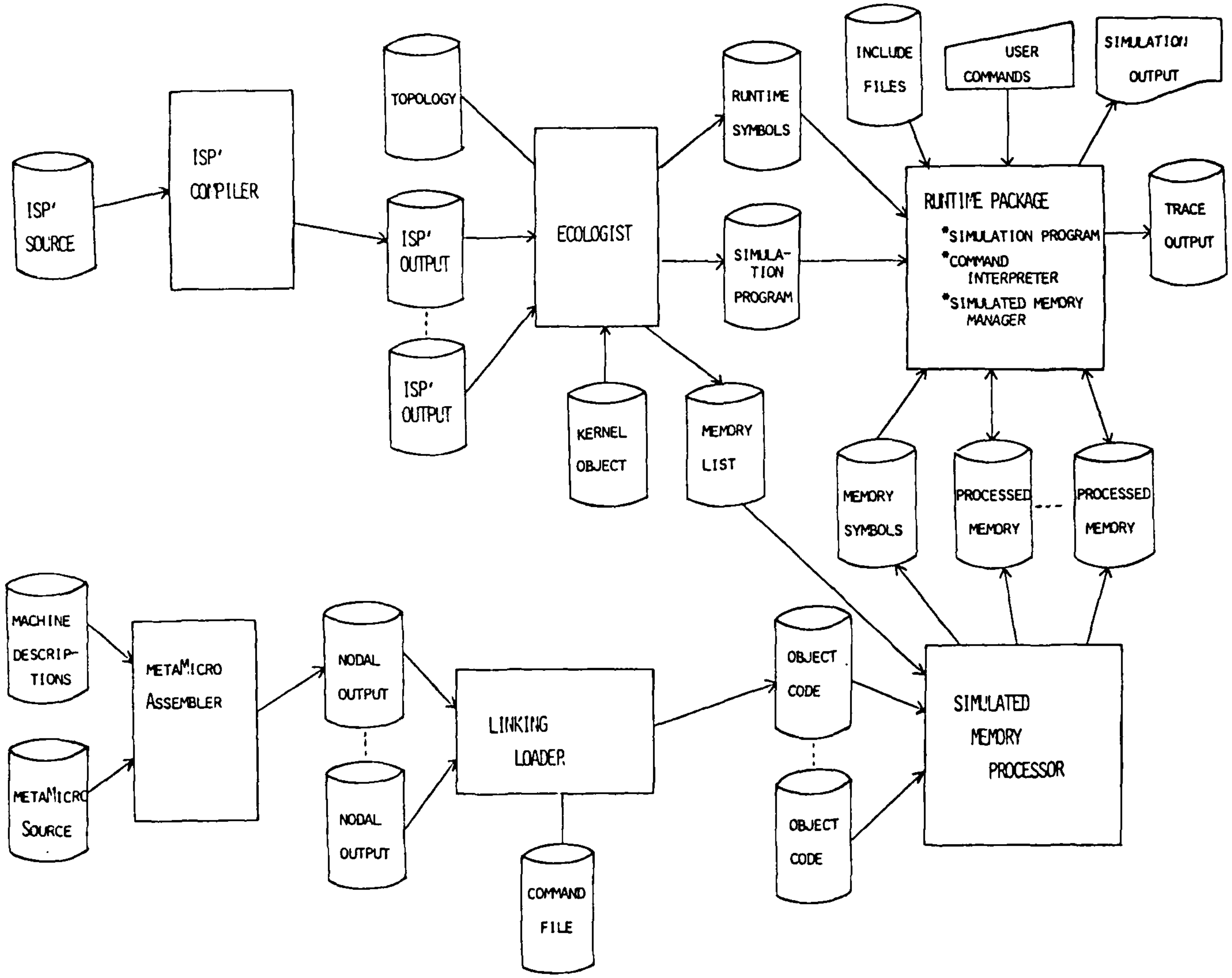


FIGURE 1: N.mPc BLOCK DIAGRAM

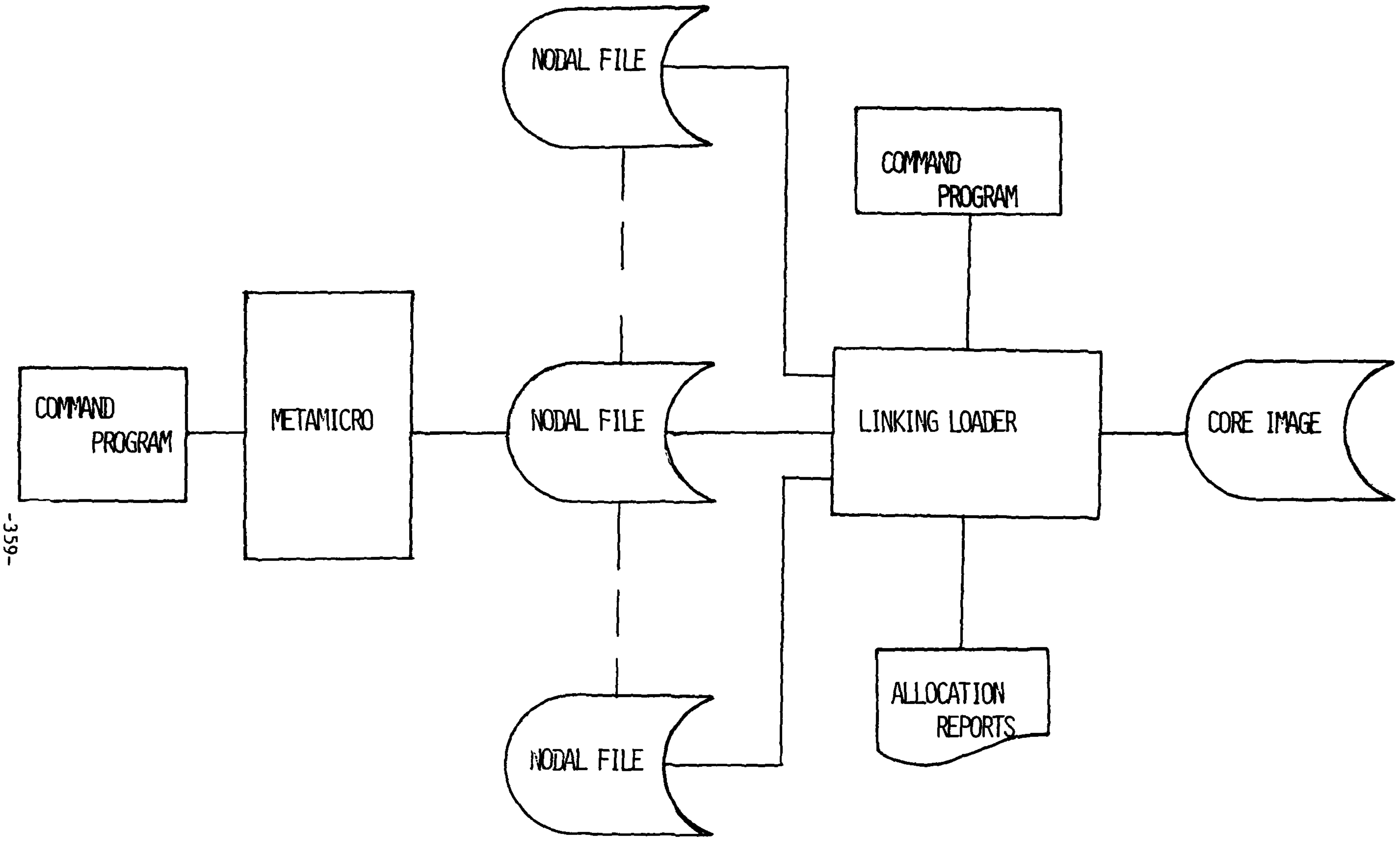


FIGURE 2: METAMICRO-LINKING LOADER SUBSYSTEM

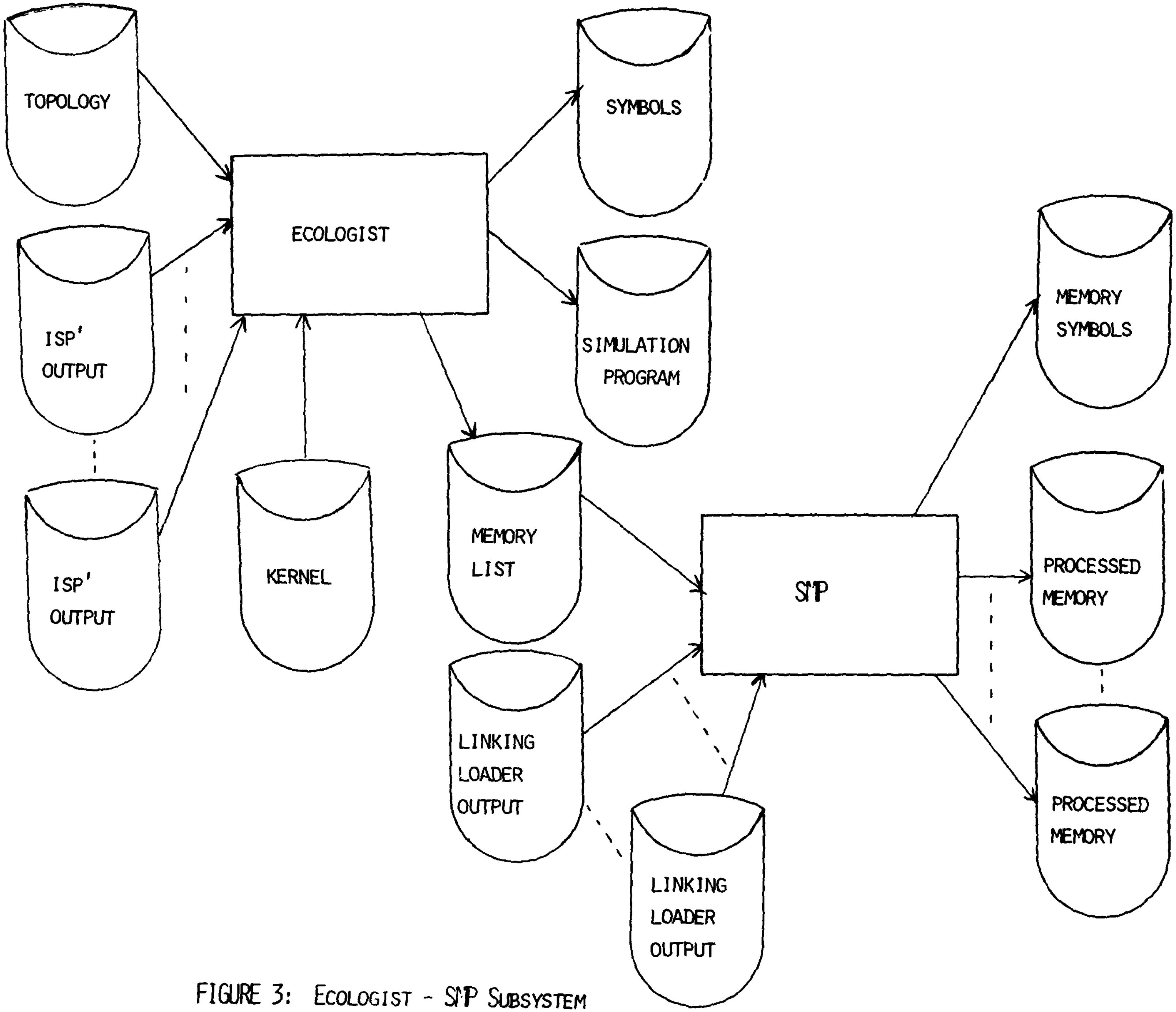


FIGURE 3: ECOLOGIST - SMP SUBSYSTEM

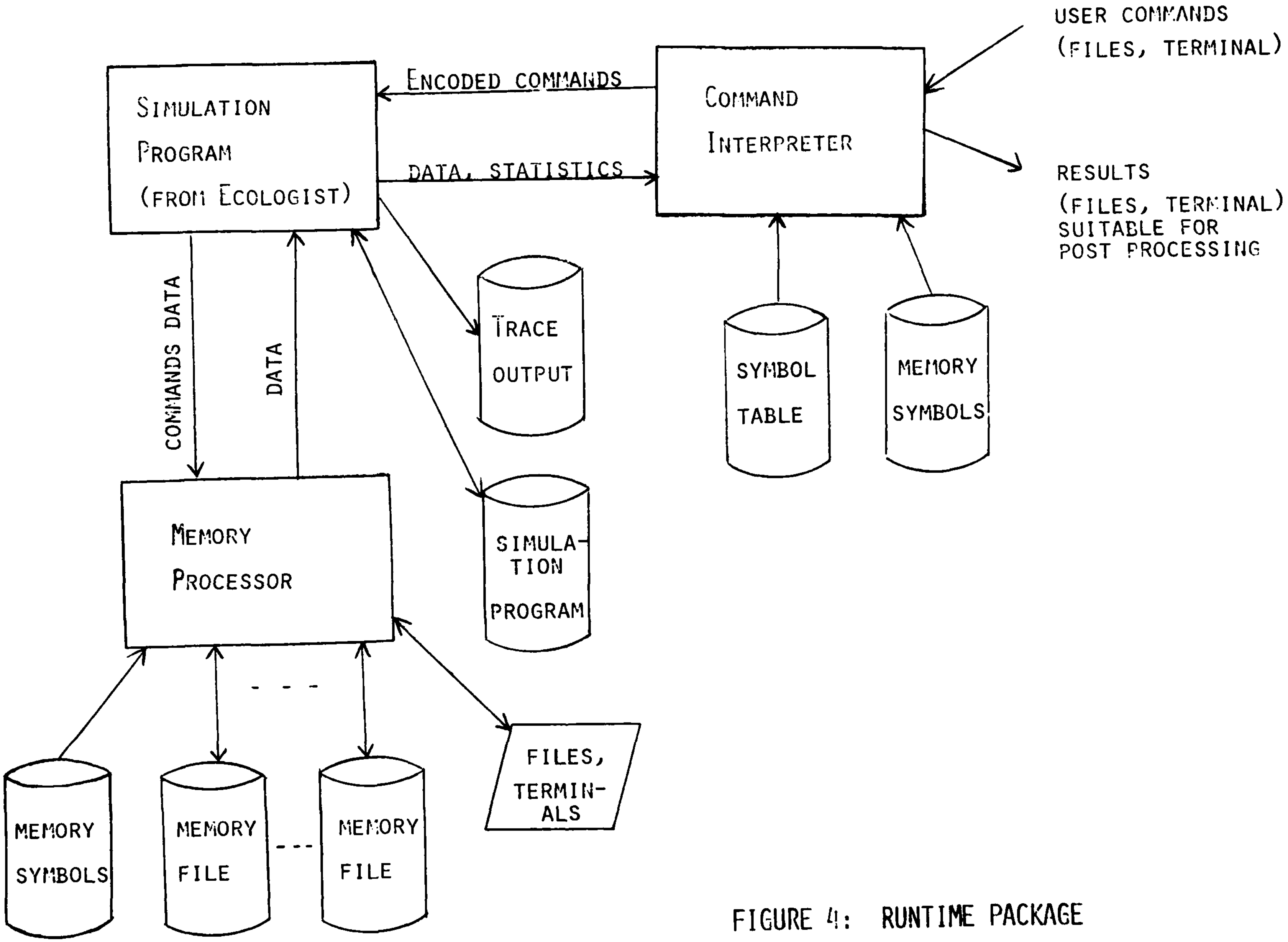


FIGURE 4: RUNTIME PACKAGE

APPENDIX A

metaMicro Description of Intel 8080

```
!       INTEL 8080 METAMICRO INPUT.      version 3
!       GREG ORDY;       Oct. 24, 1977;  i8080
!       LAST MOD: July 27, 1978;

instr   inst[3,1]<8>     $ ! three words of eight bits each
                           ! default length of instruction is 1


format  op       = inst[0]<7:6>, !main op code
        dst      = inst[0]<5:3>, !destination or op code
        src      = inst[0]<2:0>, !source or op code
        rx       = inst[0]<5:4>, !register pair
        wd1      = inst[0]<7:0>, !whole first word
        wd2      = inst[1]<7:0>, !whole second word
        wd3      = inst[2]<7:0>$ !whole third word

macro   ret      = wd1=0311 $     &,       !return unconditional
        rnz      = wd1=0300 $     &,       !return no zero (Z=0)
        rz       = wd1=0310 $     &,       !return zero (Z=1)
        rnc      = wd1=0320 $     &,       !return no carry (CY=0)
        rc       = wd1=0330 $     &,       !return carry (CY=1)
        rpo      = wd1=0340 $     &,       !return parity odd (P=0)
        rpe      = wd1=0350 $     &,       !return parity even (P=1)
        rp       = wd1=0360 $     &,       !return plus (S=0)
        rm       = wd1=0370 $     &,       !return minus (S=1)

        xchg     = wd1=0353 $     &,       !exchange DE and HL
        daa      = wd1=0047 $     &,       !decimal adjust accumulator
        rlc      = wd1=0007 $     &,       !rotate left
        rrc      = wd1=0017 $     &,       !rotate right
        ral      = wd1=0027 $     &,       !rotate left through carry
        rar      = wd1=0037 $     &,       !rotate right through carry
        cma      = wd1=0057 $     &,       !complement accumulator
        cmc      = wd1=0077 $     &,       !complement carry
        stc      = wd1=0067 $     &,       !set carry
        pchl     = wd1=0351 $     &,       !jump HL indirect
        xthl     = wd1=0343 $     &,       !exchange stack top with HL
        sphl     = wd1=0371 $     &,       !move HL to SP
        ei       = wd1=0373 $     &,       !enable interrupts
        di       = wd1=0363 $     &,       !disable interrupts
        hlt      = wd1=0166;
                   nocheck  $     &,       !HALT!
        nop      = wd1=0000 $     &,       !no operation

        in(dev)  = wd1=0333;
                   byte2(dev)     &,       !input
        out(dev)= wd1=0323;
                   byte2(dev)     &,       !output

        adi(d)   = wd1 =0306;
                   byte2(d)       &,       !add immediate
        aci(d)   = wd1=0316;
                   byte2(d)       &,       !add with C immediate
        sui(d)   = wd1=0326;
                   byte2(d)       &,       !sub immediate
```

```
sbi(d)   = wd1=0336;
           byte2(d)       &,      !sub immediate with borrow
ani(d)   = wd1=0346;
           byte2(d)       &,      !AND immediate
xri(d)   = wd1=0356;
           byte2(d)       &,      !XOR immediate
ori(d)   = wd1=0366;
           byte2(d)       &,      !OR immediate
cpi(d)   = wd1=0376;
           byte2(d)       &,      !compare immediate

ldax(RP)= wd1=0012;
           rp0(RP)        &,      !load A indirect
stax(RP)= wd1=0002;
           rp0(RP)        &,      !store A indirect

add(x)   = wd1=0200;
           sreg(x)        &,      !add reg. or mem.
adc(x)   = wd1=0210;
           sreg(x)        &,      !add with C reg. or mem.
sub(x)   = wd1=0220;
           sreg(x)        &,      !sub reg. or mem.
sbb(x)   = wd1=0230;
           sreg(x)        &,      !sub with borrow reg. or mem
inr(x)   = wd1=0004;
           dreg(x)        &,      !increment reg  or mem.
dcr(x)   = wd1=0005;
           dreg(x)        &,      !decrement reg. or mem.
ana(x)   = wd1=0240;
           sreg(x)        &,      !AND reg. or mem.
xra(x)   = wd1=0250;
           sreg(x)        &,      !XOR reg. or mem.
ora(x)   = wd1=0260;
           sreg(x)        &,      !OR  reg. or mem.
cmp(x)   = wd1=0270;
           sreg(x)        &,      !compare reg. or mem.
rst(x)   = wd1=0307;
           dreg(x)        &,      !restart

inx(RP)  = wd1=0003;
           rp1(RP) $      &,      !increment register pair
dcx(RP)  = wd1=0013;
           rp1(RP) $      &,      !decrement register pair
dad(RP)  = wd1=0011;
           rp1(RP) $      &,      !add rp to HL
push(RP)= wd1=0305;
           rp2(RP) $      &,      !push register pair
pop(RP)  = wd1=0301;
           rp2(RP) $      &,      !pop register pair

lxi(x,y)= wd1=0001;
           rp1(x);
           byte23(y)      &,      !load rp immediate

lda(x)   = wd1=0072;
```

```
          byte23(x)     &,      !load A direct
sta(x)  = wd1=0062;
          byte23(x)     &,      !store A direct
lhld(x) = wd1=0052;
          byte23(x)     &,      !load HL direct
shld(x) = wd1=0042;
          byte23(x)     &,      !store HL direct

call(x) = wd1=0315;
          byte23(x)     &,      !call unconditional
cnz(x)  = wd1=0304;
          byte23(x)     &,      !call not zero
cz(x)   = wd1=0314;
          byte23(x)     &,      !call on zero
cnc(x)  = wd1=0324;
          byte23(x)     &,      !call not carry
cc(x)   = wd1=0334;
          byte23(x)     &,      !call on carry
cpo(x)  = wd1=0344;
          byte23(x)     &,      !call parity odd
cpe(x)  = wd1=0354;
          byte23(x)     &,      !call parity even
cp(x)   = wd1=0364;
          byte23(x)     &,      !call on plus
cm(x)   = wd1=0374;
          byte23(x)     &,      !call on minus

jmp(x)  = wd1=0303;
          byte23(x)     &,      !jump unconditional
jnz(x)  = wd1=0302;
          byte23(x)     &,      !jump not zero
jz(x)   = wd1=0312;
          byte23(x)     &,      !jump on zero
jnc(x)  = wd1=0322;
          byte23(x)     &,      !jump not carry
jc(x)   = wd1=0332;
          byte23(x)     &,      !jump on carry
jpo(x)  = wd1=0342;
          byte23(x)     &,      !jump on parity odd
jpe(x)  = wd1=0352;
          byte23(x)     &,      !jump parity even
jp(x)   = wd1=0362;
          byte23(x)     &,      !jump on plus
jm(x)   = wd1=0372;
          byte23(x)     &,      !jump on minus

byte2(d)= wd2 = d;
          length = 2; $ &,      !second byte manipulation
byte23(d)=wd2 = d;
          wd3 = d^-8;
          length = 3; $ &,
                !       second and third byte manipulation

elegal(val)= (wd1=val;)iopc&,
                !       macro for illegal declarations
```

```
mov(d,s)= op = 1;
          src = s;
          dst = d; $     &,      !move

mvi(r,d)= op = 0;
          src = 6;
          dst = r;
          byte2(d)       &,      !move immediate

m       = 6     &,
a       = 7     &,
b       = 0     &,
c       = 1     &,
d       = 2     &,
e       = 3     &,
h       = 4     &,
l       = 5     &,

rim     = wd1=0040 ;$    &,      !8085 read interrupt mask
sim     = wd1=0060 ;$    &,      !8085 set  interrupt mask

dw(x)   = wd1 = x;
          wd2 = x^-8;
          length = 2;
          nocheck; $     &,      !define word pseudo instr.
db(x)   = wd1 = x;
          nocheck; $     &,      !define byte pseudo instr.

rp0(RP) = if 'RP eql "b"
          or 'RP eql "bc"
          then
          {rx = 0;$ break}
          if 'RP eql "d"
          or 'RP eql "de"
          then
          {rx = 1;$ break}
          wd1 ~ 0375; $ &,
                !       RP must be: b,bc,d, or de

rp1(RP) = if 'RP eql "b"
          or 'RP eql "bc"
          then
          {rx = 0 break}
          if 'RP eql "d"
          or 'RP eql "de"
          then
          {rx = 1 break}
          if 'RP eql "h"
          or 'RP eql "hl"
          then
          {rx = 2 break}
          if 'RP eql "sp"
          then
          {rx = 3 break}
          wd1 ~ 0335    &,
                !       RP must be: b,bc,d,de,h,hl, or sp

rp2(RP) = if 'RP eql "b"
          or 'RP eql "bc"
          then
          {rx = 0 break}
          if 'RP eql "d"
          or 'RP eql "de"
          then
          {rx = 1 break}
          if 'RP eql "h"
          or 'RP eql "hl"
          then
          {rx = 2 break}
          if 'RP eql "psw"
          then
          {rx = 3 break}   .
          wd1 ~ 0331    &,
                !       RP must be: b,bc,d,de,h,hl or psw

sreg(x) = if r8(x)
          then { src=x;$ break}
          wd1 ~ 0355;$  &,      !check that src is a registe

dreg(x) = if r8(x)
          then { dst=x;$ break}
          wd1 ~ 0355;$  &,      !check that dst is a registe

r8(x)   = (('x geq "a"
             and 'x leq "e")
          or
           ('x eql "h"
            or 'x eql "l"
            or 'x eql "m")) &$
                !       evaluate parameter as a register

This section contains the 8080 and 8085 illegal op codes


bind    iopc    "ILLEGAL OP CODE FORMED",
        bcde    "REGISTER PAIR MAY ONLY BE bc OR de",
        rp1c    "REGISTER PAIR MAY ONLY BE: b,d,h,sp",
        rp2c    "REGISTER PAIR MAY ONLY BE: b,d,h,psw",
        mbrg    "OPERAND MUST BE A REGISTER" $

illegal elegal(0010),elegal(0020),elegal(0030),
        elegal(0050),elegal(0070),
        elegal(0313),elegal(0166),
        (wd1 = 0375) bcde,
        (wd1 = 0335) rp1c,
        (wd1 = 0331) rp2c,
        (wd1 = 0355) mbrg $
```

APPENDIX B

8080 Interpreter Program

```
%*************************************************************%
%                                                             %
%       Machine:            Intel 8080                        %
%                                                             %
%*************************************************************%

instr   inst[3,1]<8>$

format  op      =       inst[0]<7:6>,
        dst     =       inst[0]<5:3>,
        src     =       inst[0]<2:0>,
        rx      =       inst[0]<5:4>,
        wd1     =       inst[0]<7:0>,
        wd2     =       inst[1]<7:0>,
        wd3     =       inst[2]<7:0>$

mode
        case  length eql 3:
                wd2 ~ (wd3^8) + wd2 + address$
                wd3 ~ ((wd3^8) + wd2 + address)^-8$
                break$
        esac,

        default:
                wd1 ~ (wd2^8) + wd1 + address$
                wd2 ~ ((wd2^8) + wd1 + address)^-8$
                break$
        esac$


space   <0x0000:0x0010>,
        <0x0020:0x0050> $

transfer {      new
                wd1 = 0303$
                wd2 = address$
                wd3 = address ^ -8$
                length = 3$     }
```

APPENDIX C

ISP' Description of 8080

```
!
!       i8080.isp - isp' 8080 program, with external memory,
!       no interrupts, and full timing information
!



macro   BYTE     =        8&,
        WORD     =        16&,
        HIBYTE   =        15:8&,
        LOBYTE   =        7:0&,
        CYCLE    =        delay(1)&,
        X9(val)  =        val ext 9&,
        X5(val)  =        (val ext 4) ext 5&,
        X17(val) =        val ext 17&;

state   a<BYTE>,                  ! accumulator
        psw<BYTE>,                ! processor status word
        bc<WORD>,                 ! b and c register pair
        de<WORD>,                 ! d and e register pair
        hl<WORD>,                 ! h and l register pair
        sp<WORD>,                 ! stack pointer
        pc<WORD>,                 ! program counter
        ir<BYTE>;                 ! instruction register

port    abus<WORD>,               ! address bus
        dbus<BYTE>,               ! data bus
        memr(1),                  ! inverted memory read
        memw(1),                  ! inverted memory write
        ior(1),                   ! inverted port read
        iow(1);                   ! inverted port write

format  b        =        bc<HIBYTE>,
        c        =        bc<LOBYTE>,
        d        =        de<HIBYTE>,
        e        =        de<LOBYTE>,
        h        =        hl<HIBYTE>,
        l        =        hl<LOBYTE>,
        cy       =        psw<0>,
        p        =        psw<2>,
        ac       =        psw<4>,
        z        =        psw<6>,
        s        =        psw<7>,
        src      =        ir<2:0>,
        dst      =        ir<5:3>,
        op       =        ir<7:6>,
        ccc      =        ir<5:3>,
        rp       =        ir<5:4>;


/************************************************/
/*   geturd - 3 cycle memory read, with         */
/*   address specified                          */
/************************************************/
```

```
geturd(addr<WORD>)<BYTE> :=
        (abus = addr;
        CYCLE;
        memr = 0;
        CYCLE;
        geturd = dbus;
        CYCLE;
        abus = 0;
        memr = 1;
        )


/************************************************/
/*   data8 - 3 cycle memory read, uses pc for   */
/*   address, and increments pc immediately     */
/************************************************/

data8<BYTE> :=
        (data8 = geturd(pc);
        pc = pc+1
        )


/************************************************/
/*   data16 - 6 cycle memory read, uses pc for  */
/*   address, and adds 2 to pc immediately      */
/************************************************/

data16<WORD> :=
        (data16 = geturd(pc+1) concat geturd(pc);
        pc = pc+2
        )


/************************************************/
/*   storeurd - 3 cycle memory write, with      */
/*   address specified                          */
/************************************************/

storeurd(addr<WORD>,val<BYTE>) :=
        (abus = addr;
        dbus = val;
        CYCLE;
        memw = 0;
        CYCLE;
        memw = 1;
        CYCLE;
        abus = 0;
        dbus = 0;
        )
```

o
o
o

```
/************************************************/
/*   main program - perform instruction         */
/*   decoding, and execute instructions         */
/************************************************/

main := (
        state   temp8<BYTE>,
                temp16<WORD>,
                alu16save<17>;

ir = data8;                                     ! fetch
next;


case op
   0: case ir<3:0>
      1: storerp(data16)                        ! LXI
      2: case ir<5:4>
         0,1: storewrd(getrp,a)                 ! STAX
         2: (temp16 = data16; next              ! SHLD
            storewrd(temp16,l);
            storewrd(temp16+1,h)
            )
         3: storewrd(data16,a)                  ! STA
         esac
      3: (storerp(getrp+1);                     ! INX
         CYCLE
         )
      4,014: (temp8 = getdst; next              ! INR
            storedst(alu(X9(temp8) + 1));
            cac(X5(temp8) +1);
            CYCLE
            )
      5,015: (temp8 = getdst; next              ! DCR
            storedst(alu(X9(temp8) -1));
            cac(X5(temp8) -1);
            CYCLE
            )
      6,016: storedst(data8)                    ! MVI
      7: case ir<5:4>
         0: (a = a *:rotate 1; cy = a<7>)       ! RLC
         1: (a = a concat cy ; cy = a<7>)       ! RAL
         2: (if (a<3:0> ext 5) gtr 9 or ac      ! DAA
               (a = a+6 ; ac = 1)
            else
               ac = 0; next
            if (a<7:4> ext 5) gtr 9 or cy
               (a = a+6 ; cy = 1)
            else
               cy = 0; next
            temp8 = alu(a)
            )
         3: cy = 1                              ! STC
         esac
```

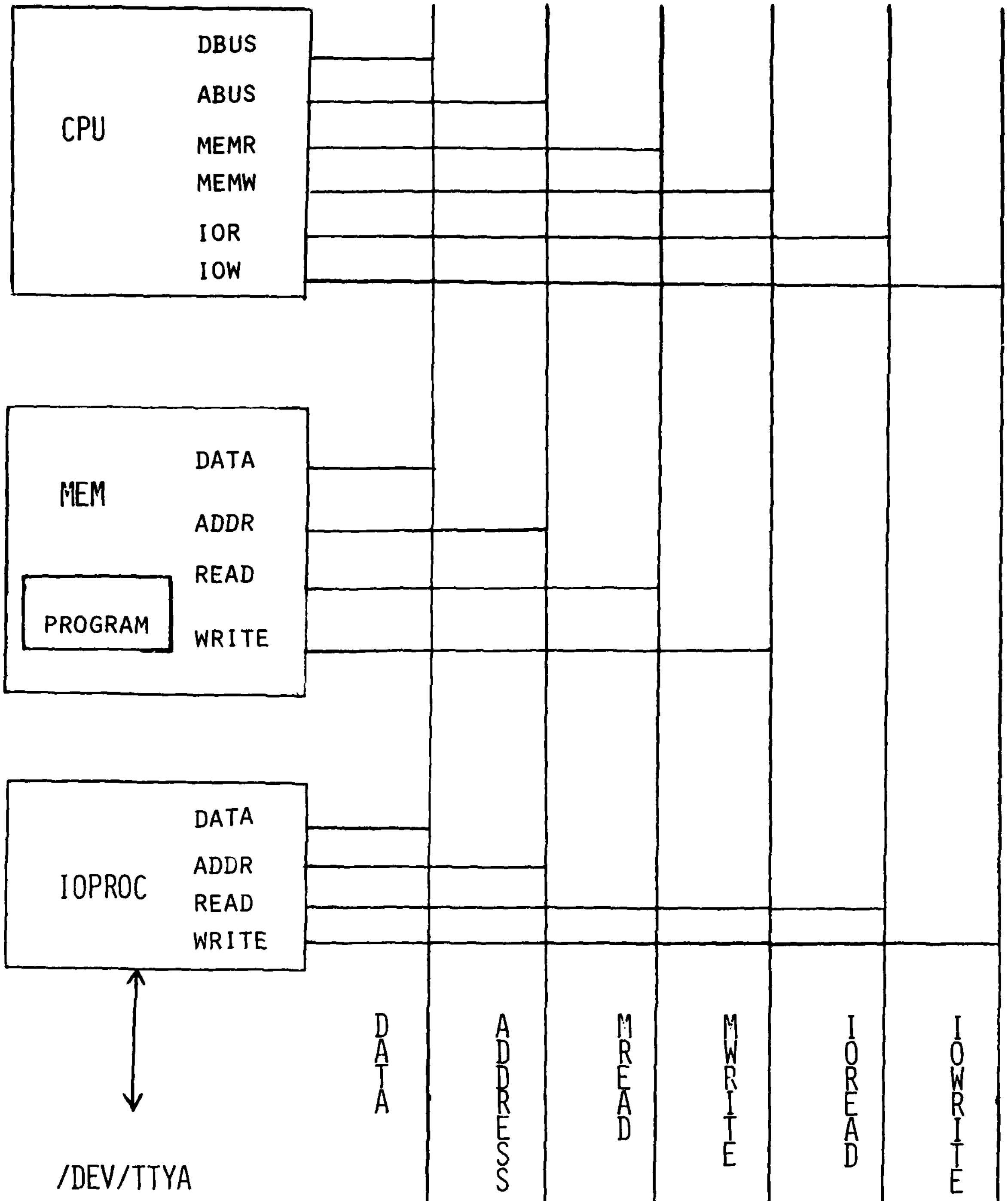
```
          o
          o
          o
   0: pc = data16                         ! JMP
   1: ;                                   ! no-op
   2: (abus = data8 ext 16;               ! OUT
      dbus = a;
      CYCLE;
      iow = 0;
      CYCLE;
      iow = 1;
      CYCLE;
      abus = 0;
      dbus = 0;
      )
   3: (abus = data8 ext 16;               ! IN
      CYCLE;
      ior = 0;
      CYCLE;
      a = dbus;
      ior = 1;
      abus = 0;
      CYCLE;
      )
   4: (hl = pop ; push(hl) ; CYCLE )      ! XTHL
   5: (de = hl ; hl = de)                 ! XCHG
   6: ;                                   ! DI
   7: ;                                   ! EI
   esac
4: (temp16 = data16; next ! Conditional Call
   if (cctest)
      (push(pc);
      pc = temp16
      );
   CYCLE
   )
5: case ir<5:3>
   0,2,4: (push(getrp);                   ! PUSH
      CYCLE
      )
   1: (pc = data16 ; push(pc) ; CYCLE)    ! CALL
   6: (push(a concat psw);        ! PUSH psw
      CYCLE
      )
   3,5,7: ;                               ! no-op
   esac
6: case ir<5:3>
   0: (temp8 = data8; next                ! ADI
      a = aluc(X9(a) + X9(temp8));
      cac(X5(a) + X5(temp8))
      )
   1: (temp8 = data8; next                ! ACI
            cac(X5(a) + X5(temp8) + X5(cy));
            a = aluc(X9(a) + X9(temp8) + X9(cy))
            )
         2: (temp8 = data8; next                 ! SUI
            a = aluc(X9(a) - X9(temp8));
            cac(X5(a) - X5(temp8))
            )
         3: (temp8 = data8; next                 ! SBI
            cac(X5(a) - X5(temp8) - X5(cy));
            a = aluc(X9(a) - X9(temp8) - X9(cy))
            )
         4: (a=aluc(X9(a) and X9(data8)) ; ac=0)! ANI
         5: (a=aluc(X9(a) xor X9(data8)) ; ac=0)! XRI
         6: (a=aluc(X9(a) or X9(data8)) ; ac=0) ! ORI
         7: (temp8 = data8; next                 ! CPI
            temp8 = aluc(X9(a) - X9(temp8));
            cac(X5(a) - X5(temp8))
            )
         esac
      7: (push(pc);                              ! RST
         pc = ir<5:3> ext 16 *:arith 3;
         CYCLE
         )
      esac
   esac
)
```

APPENDIX D

Topology Example

```
!       Topology for small 8080 system

signal  data(8),            ! data bus
        address(16),        ! address bus
        mread,              ! memory read control line
        mwrite,             ! memory write control line
        ioread,             ! i/o read control line
        iowrite;            ! i/o write control line

!       declare main processor (8080)

processor       cpu = i8080.sim;

time delay 500 ns;          !run with 2 Mhz clock

connections     dbus = data,
                abus = address,
                memr = mread,
                memw = mwrite,
                ior  = ioread,
                iow  = iowrite;


!       declare memory module

processor       mem = memmodule.sim;

connections     data  = data,
                addr  = address,
                read  = mread,
                write = mwrite;

!       use linking loader output file, "program" as
!       memory contents

initial         Memory = program;


!       declare i/o processor, connected to the
!       UNIX device, "/dev/ttya"

processor       ioproc = ioproc.sim;

connections     data  = data,
                addr  = address,
                read  = ioread,
                write = iowrite;

!       tie isp' memory, tty, to the raw memory, /dev/ttya

initial         tty = (/dev/ttya);
```

BLOCK DIAGRAM OF 8080 SYSTEM



INTEROPERABILITY SESSION

Dwaine B. Huewe
CENTACS

INTEROPERABILITY SESSION

SESSION CHAIRPERSON: Dwaine B. Huewe

Director,
Center for Systems Engineering and Integration
CENTACS

SESSION SUMMARY

Interoperability was defined in the opening paragraphs. Army organization to develop and implement interoperability was presented. This included an explanation of the complex relationships of operational requirements and technology.

The background discussion included explanations of Joint Interoperability Tactical Command and Control Systems (JINTACCS), the intra-Army program, and support of NATO interoperations. Specific problems were identified.

Problems, once identified, require solution. An approach to solving interoperational problems was developed. Both technical and management factors were considered.

Interoperability problems and their solutions impact on other activities; i.e., doctrine, operational requirements, standards, communication technology, the specifications of individual systems, and the architecture of the overall tactical Command, Control, Communications and Intelligence (C^3I) systems. This was analyzed.

Workable software is the key to successful interoperations. The major portion of the paper covered this subject. Included were fundamental concepts of such basics as man-machine relationships and the technology of computer programs. Such subjects as firmware, common modules, common algorithms, and message formats, were discussed. The importance to interoperations of programming in high order languages was brought out.

On a system-wide basis -- the overall C^3I tactical system -- there are problems of system test, overall configuration management, and C^3I system management. These problems were analyzed. Technical and management solutions were presented.

INTEROPERABILITY
TACTICAL AUTOMATED SYSTEMS
A CHALLENGE

Dwaine B. Huewe

Director, Center for Systems
Engineering and Integration
Fort Monmouth, New Jersey

SUMMARY

The central theme of the paper is the challenge to the R&D software community to provide the necessary Management Control and visibility to assure interoperability throughout the life cycle of the system. The developer is challenged to use initiative in the development of new innovations and incorporate these new innovations in the software design.

Major topics include:

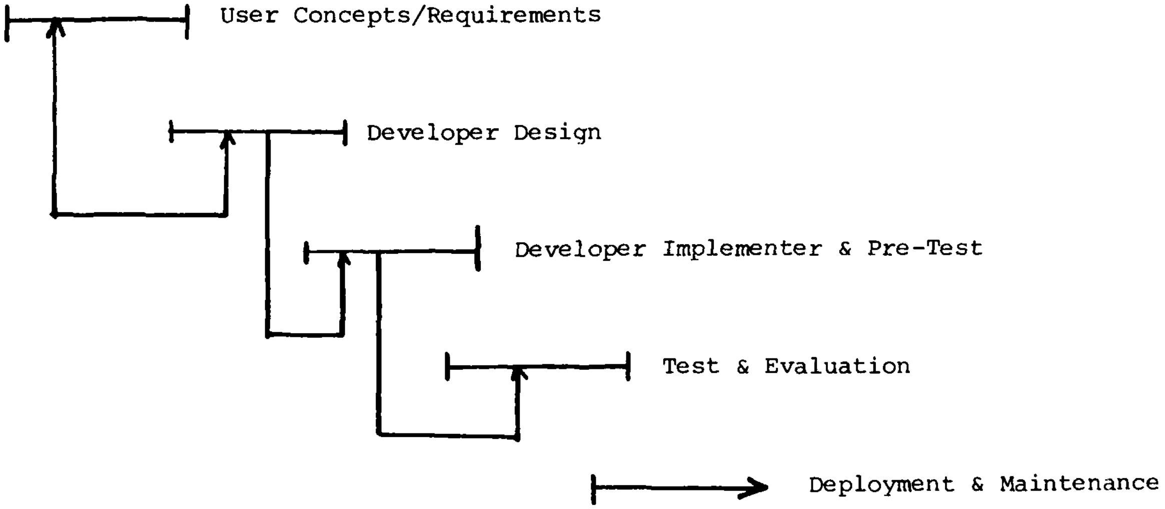
- Army Interoperability Program
- Approach to Interoperability
- Software Challenges
- Management Issues
- Challenge to the Developer

ARMY INTEROPERABILITY PROGRAM

In the past, Army systems have been independently developed principally to meet a user defined mission. In systems where a specific interoperability requirement was defined, interoperability for that requirement was provided. Most requirements were nebulous or undefined. Therefore, no coordination was effected between developers. These independent developments have contributed to the complexity of the battlefield, wasteful use of the spectrum, equipment proliferation, and unnecessary costs. Further, we have not fully capitalized upon the inherent capabilities of the system. Today, we are trying to bring both automated tactical systems and communication systems to one system architecture or frame work with some degree of orderliness to achieve the system interoperability objectives.

To provide effective, centralized management, the Center for Systems Engineering and Integration (CENSEI), Figure 1, was established with CORADCOM as the Army focal point for C^3 System Engineering and managing, coordinating, and implementing interoperability requirements of Army Command, Control, Communications and Intelligence (C^3-1). Currently the Army is involved in two major interoperability efforts - JINTACCS (Joint Interoperability of Tactical Command and Control Systems) and Intra-Army, with Army interoperability within NATO as a new initiative.

CENSEI views the life cycles of any interoperability programs as paralleling the life cycle of a system. That is, the user concepts and requirements are established; developer trade-offs, constraints, and capabilities are proposed, coordinated and designed for meeting those user requirements; developer implementation and preliminary technical testing of the requirements; test agency evaluation and analysis; and finally, deployment and maintenance. Figure 2 outlines the chronology of these events. However, it does not show the most important aspect of interoperability program evolution - - the constructure coordination necessary among all the diverse organizations involved.



In an interoperability program such as JINTACCS, each organization - user, developer and tester - is made up of representatives of each service. Therefore, the JINTACCS user "organization" will have Army, Navy, Marine, Air Force, and DIA/NSA users convene periodically to work out joint interoperability requirements. Similarly, the develooer "organization" made up of the service representatives will then meet to work out the implementation. The test "organization" is somewhat more complex in tnat all services test representatives/agencies perform testing under a single joint test organization - the Joint Interface Test Force (JITF) - that plans and coordinates the joint testing.

Needless to say, such interoperability programs are replete with the traditional problems associated with a single system's evolution. Such problems as inadequate user requirements, technology improvements that, if determine desirable, restructure the design and implementation, overly optimistic scheduling and funding based on "all success" planning, and so on, plagues interoperability programs just as they do single system programs. However, the adverse effects are multiplied exponentially because there are many systems involved and because there are additional problems not related to the fundamental mission. The additional problems are primarily political (which organization or service is "in charge"), economic, (who modifies his system to be compatible with someone else's system determines who pays for and takes the risks of modification) and technical (which of many currently used standards for messages, data elements, or communications will be the best compromises to use for interfaces or are new standards needed).

INTRA-ARMY'S APPROACH TO INTEROPERABILITY

Interoperability development is based upon the following documents;

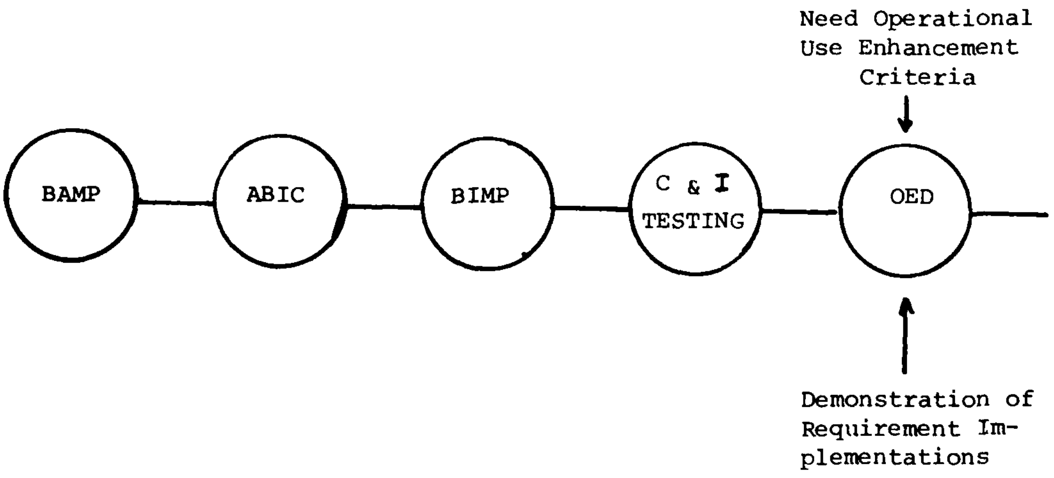
- The Battlefield Automation Management Plan
- Automatic Battlefield Interface Concepts
- Battlefield Interoperability Management Plan
- Technical Interface Design Plan

The relationship and description of each document is summarized in the chart below:

BAMP	ABIC	BIMP	TIDP
Evaluates all current and future automated systems Methodology: Design Analysis Performance Eval. Cost Evaluation Pay-Off Evaluation Recommends: Accelerate Continue Terminate Modify Rejustify	States the requirements for automated system interop Describes: Who What Where Based On: BAMP Shortfalls JINTACCS Rqts. STANAGS NATO Requirements Provides: Interop Rqts. Resources	The Management structure and plans for the development & implementation of ABIC interfaces Contains: Responsibilities Authorities Relationships Resources Schedules	Describes the design and engineering requirements to accomplish the ABIC interfaces Contains: MSG Formats Data Lines Data Elements Security Commo CONOPS Protocols

Unlike past interoperability programs, resources and funding authority is provided to implement interoperability requirements.

The major events for Intra-Army Interoperability which is similar to the joint service methodology is as depicted below:


Need Operational Use Enhancement Criteria
BAMP
ABIC
BIMP
C & I TESTING
OED
Demonstration of Requirement Implementations

The first event has been completed.

CHALLENGE

The Center for Systems Engineering Directorate has identified significant challenges in the Interoperability arena. The interoperability problem is complex impacting upon every system in the battlefield. We currently do not have solutions to all the interoperability problems. The final and optimum solution will be provided by best efforts of industry and government personnel. The interoperability challenge provides lots of opportunity for "innovative thinkers and doers".

The impact of interoperability involves every segment of a system such as system design, interface characteristics, computer programs, data base, and communications. Externally it impacts upon the operator, the communication media, management and, because of its evolutionary nature, doctrine. The impact upon the management structure is great because it forces a higher degree of centralization and control due to involvement of two or more systems.

Some of the key software interoperability considerations are:

- Man/Machine Interface
- Software versus Firmware
- Message Generation
- Standards
- Security
- Training
- Test
- CONOPS
- Survivability

The above considerations are not all inclusive, but offer significant challenges to the software developer.

MAN/MACHINE INTERFACE AND SOFTWARE/FIRMWARE CHALLENGE

The first two considerations - Man/Machine Interface and Software/Firmware, are related in that the software/firmware offers a solution to the man/machine interface. The man/machine interface goals as they relate to interoperability are:

- Standard Operator Procedure
- Interactive Man/Machine Interface
- Common Software/Firmware Control
- Imbedded Operator Training Routines
- Common Input/Output Devices
- Multi-Applications

In considering the software/firmware trade-offs, the developer must consider commonality of software, programs, transportability to other systems, commonality of modules/algorithms, etc., yet retain the flexibility to meet all requirements. The trend has been toward flexible, adaptable software/hardware, easy to change to meet the requirements.

The challenge below reverses that trend toward more standardization using ROM's to provide an even greater degree of control over change. Hence, the challenge is to provide:

- Common firmware to provide standard picture for alphanumerics and graphics,
- Common keyboard interaction

The efforts to meet the challenge to-date have been limited to standard symbology for U.S. and NATO. The use of a standard keyboard, display controls, have been demonstrated by the programmable display driven keyboard of the Digital Message Device and the Tactical Computer System. The need exists to standardize keyboard and controls across the systems.

Message Generation Challenges

The battle continues with respect to fixed versus variable formats, bit oriented and character oriented message, and definition of the protocols, etc. The formats will probably be a combination of fixed, variable, bit oriented, and character oriented messages. However, it is a goal to standardize all protocols and formats to the maximum extent possible, resulting in standard message format generation routines. It may not be practical to provide a standard generation routine due to the large number of required formats for independent systems. The challenge is to provide flexible message generation capabilities to the lowest design level without impacting upon own and other systems communication processors, files and application programs.

Standard Challenge

The use of standards are not new to the Army, but the proliferation of standards has reached a point of saturation. The developer or user has manv options from which to choose. The options selected may be completely incompatible with other systems if selected independently by the system developer. This trend will slowly be reversed to more rigid standards, impacting upon development standards, performance standards training, design, documentation and scenarios. We need to review our practices to provide assurance that interoperability will be achieved. It is our goal to provide a common base for interoperability across all systems including design of standard software/firmware modules, scenario's for interoperability testing, on-line and off-line software training routines, standard documentation, and procedures.

What is being done today for development of standardization? Not enough. Standards is a long evolutionary process. Message standards, and communications standards for U.S. and NATO are in the process of being developed. Progress of software interoperability standard development is now known, therefore, it is assumed that software interoperability standards are non-existent. The challenge is to develop, distribute, and maintain current standards and distribute changes responsively to all affected activities simultaneously. Remember a standard is a known measurable point of departure.

Training Challenge

The training objectives for interoperability include the following:

- Continuous on-the-job training and operations on the same system.
- On-line/off-line training routines/exercises.
- Standard documentation.
- Common training base.
- Standard training procedures.
- Scenario's.

The training objectives probably provide the most complex challenges of all. Simply, we desire to provide the operator interoperability training on the same system that he used for operations using on-line and off-line training exercises, in garrision and in the field. Secondly, using a common training base, standard documentation, procedures and scenario's produces a trained operator that can be moved from one similar function to another similar function of another system with no training. No known progress has been made to achieve this objective.

Security Challenge

Security is not new to the Army systems. We are all familiar with such terms as system security, TEMPEST, accidental desclosure of information, deliberate penetration, passive infiltration, multi-level security, security destruct procedures, physical security. When security is considered in the view of interoperability, a new dimension of security is added. The complexity is increased expanentially, due to the number of systems and levels of security withing each system. In addition, the levels of security for Joint, Army, and NATO are varied and may cause severe impact upon systems in terms of design, schedule, and dollars. Interoperability provides new avenues to infiltrate a computer system and compromise the system. The interoperability impacts upon security doctrine and regulation in that new security operations procedures may result in new controls for multi-system and new guidelines for multi-level security may be required. The challenge is to provide a common security module or package that will provide adequate multi-level security for Joint - Army - NATO interoperability in a hostile environment. No progress has been made in this area except toward total encryption.

Test Challenge

The interoperability test philosophy follows the philosophy that exists today for a system coordinated test program. The techniques and methodology for collection of data may vary due to involvement of multi-systems. Some key considerations for interoperability tests are:

- Develop test Performance Criteria Concurrent with System Requirements
- Test at all Levels
  - Sub-Module
  - Module
  - Integrated Software
  - Integrated System
  - Operational Test Demonstration
- Exercise all Interfaces
- Imbedded Test Routines
- *Maximum use of Simulation Techniques*
- On-Line Data Collection Routines
- On-Line/Off-Line Data Reduction Routines
- Standard Scenario

The software challenge is to what level interoperability is affected and level of testing to be performed. Does a change in one module of the software program affect other modules within all systems? How is the impact determined? How is interoperability validated?

The progress made in this area has been limited to compatibility and interoperability testing. Compatibility exists when two systems can exchange information. Interoperability is achieved when two or more systems can exchange information and use the informaion exchanged. The compatibility and integration testing is a single thread test which does not measure the impact upon system performance. The Teleprocessing Design Center, located at Fort Monmouth, New Jersey, provides a multi-system capability which allows performance testing of tactical control systems by the use of emulation. The performance monitoring information (reduced to usable data) becomes the basis for performance evaluation of tactical systems.

The micro-programmable multi-processor computer system in the TDC has the necessary hardware, firmware, and software to emulate total computer systems. The need exists for the system developer to have the necessary tools to test system interoperability at the necessary level required to validate interoperability through the use of software routines, simulation techniques, data collection/reduction, without other interoperable systems that are not available for test.

Continuity of Operations Survivability (CONOPS) Challenge

Continuity of Operations and Survivability are not new terms to the software developer. He is aware of the considerations for CONOPS and survivability:

- Back-up Capability
    - Common Software
    - Common Data Base
    - Update Procedures
    - Gradual Degradation
- Maintenance of Program
- Recovery from Catastrophic Failure due to:
    - Jamming
    - EMP
    - Human Error
    - Sabotage

Although a total awareness of the above exists for CONOPS and survivability, these programs have never reached the ultimate objective. This ultimate objective is complicated further by the number of Joint, NATO, and the individual service interoperability requirements that have created a dependancy upon other systems for real time data exchange.

The challenge is to provide distributed, essential information to more than one node of a distributed system so any node can collect essential data from other nodes to perform the function of the destroyed node.

INTEROPERABILITY MANAGEMENT AND CHALLENGE

Traditionally the Army Project Manager has served as the system developer in a more independent role. The Manager as well as the user and the tester must assume an expanded role to include interoperability. Nevertheless, the trend in interoperability Management will be toward more centralized authority due to multi-system/Joint Service/NATO involvement with execution of the interoperability decentralized at lower levels of command.

Perhaps the largest impact is upon Configuration Management. Configuration, to be responsive must be decentralized to the lowest possible level. The trend, because of the large number of systems involved, is to centralize configuration management at upper management levels. We need to review our management structure, our Configuration Management Plans, the change cycles to determine their adequacy to satisfy interoperability objectives. How much control should be exercised upon the system? What level is changes approved? How is a determination made that a change affects or does not affect interoperability of our own and other systems?

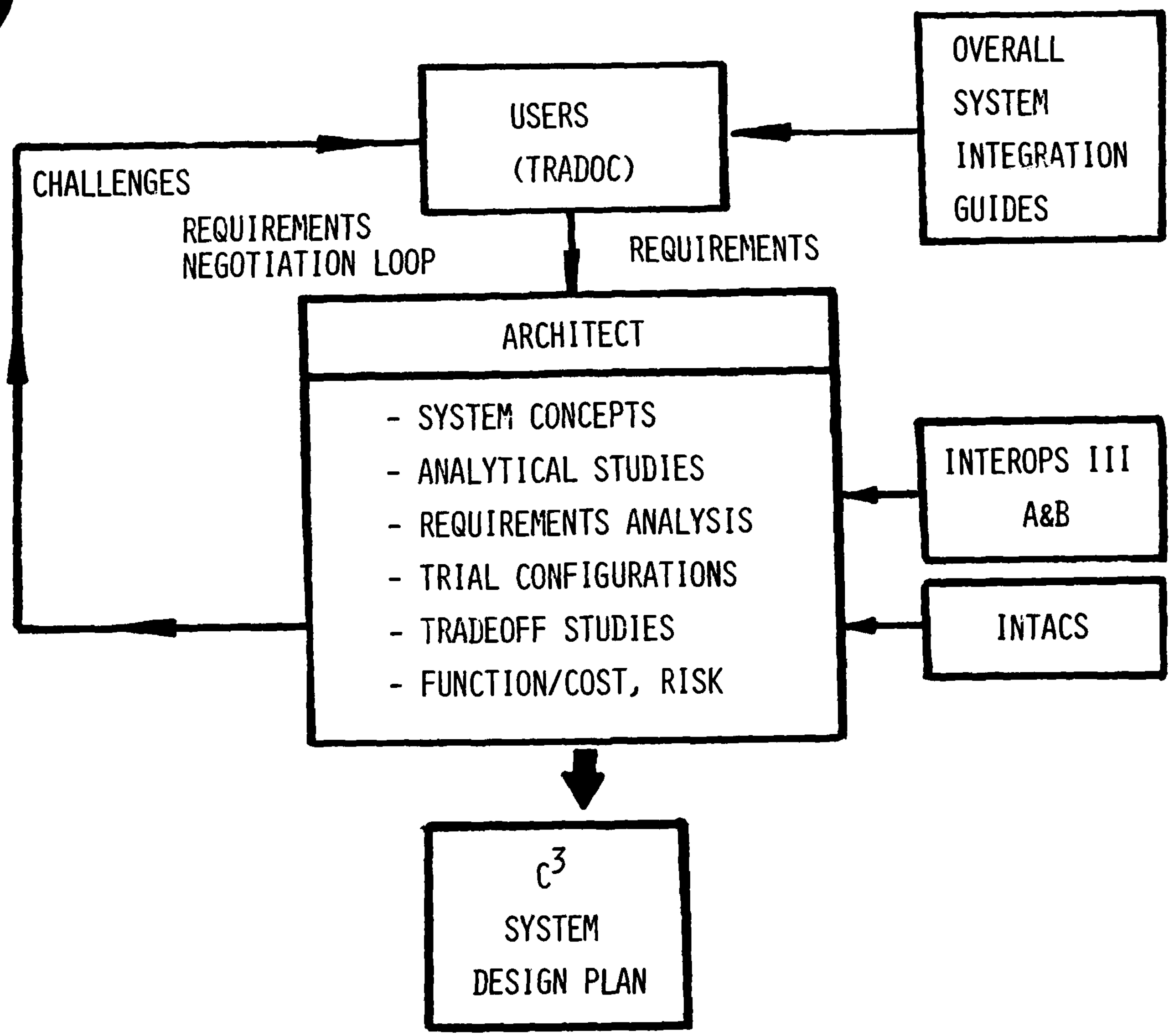
It may lead to more standard software development procedures, detail documenation standards, and new performance standards. The challenge is to provide a real time destributed management system throughout the interoperability community.

The full impact of intercperability upon management has not been realized. Because the full impact has not been realized, progress to meet this challenge has been slow. The ARPANET and other computer networks has potential for providing this real time destributed management system.

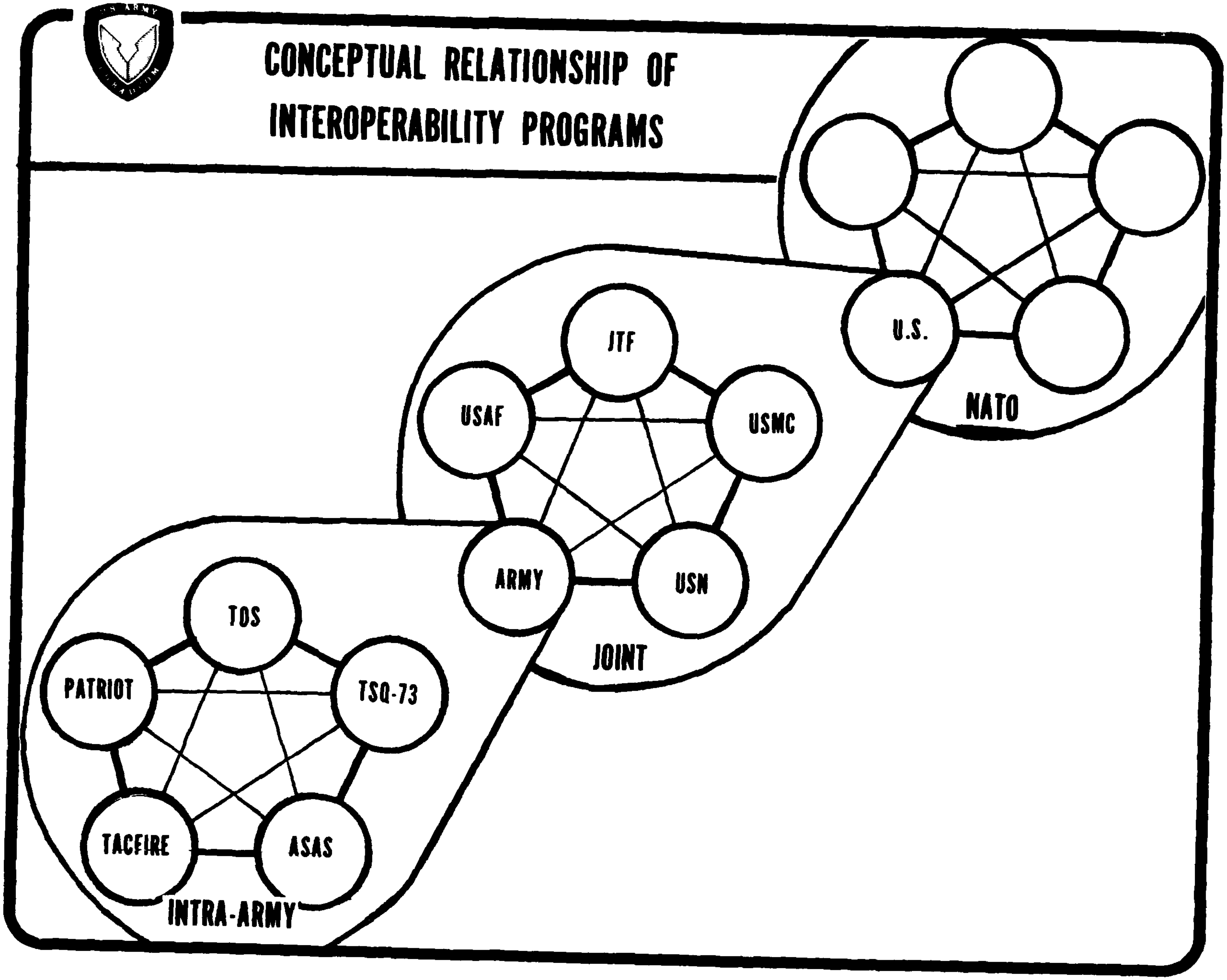
CHALLENGE TO THE DEVELOPER

We do not have solutions to all software interoperability problems. Interoperability needs help that must come from the best in government and industry. As stated initially in the challenge, interoperability provides lots of opportunity for innovative thinkers and doers.

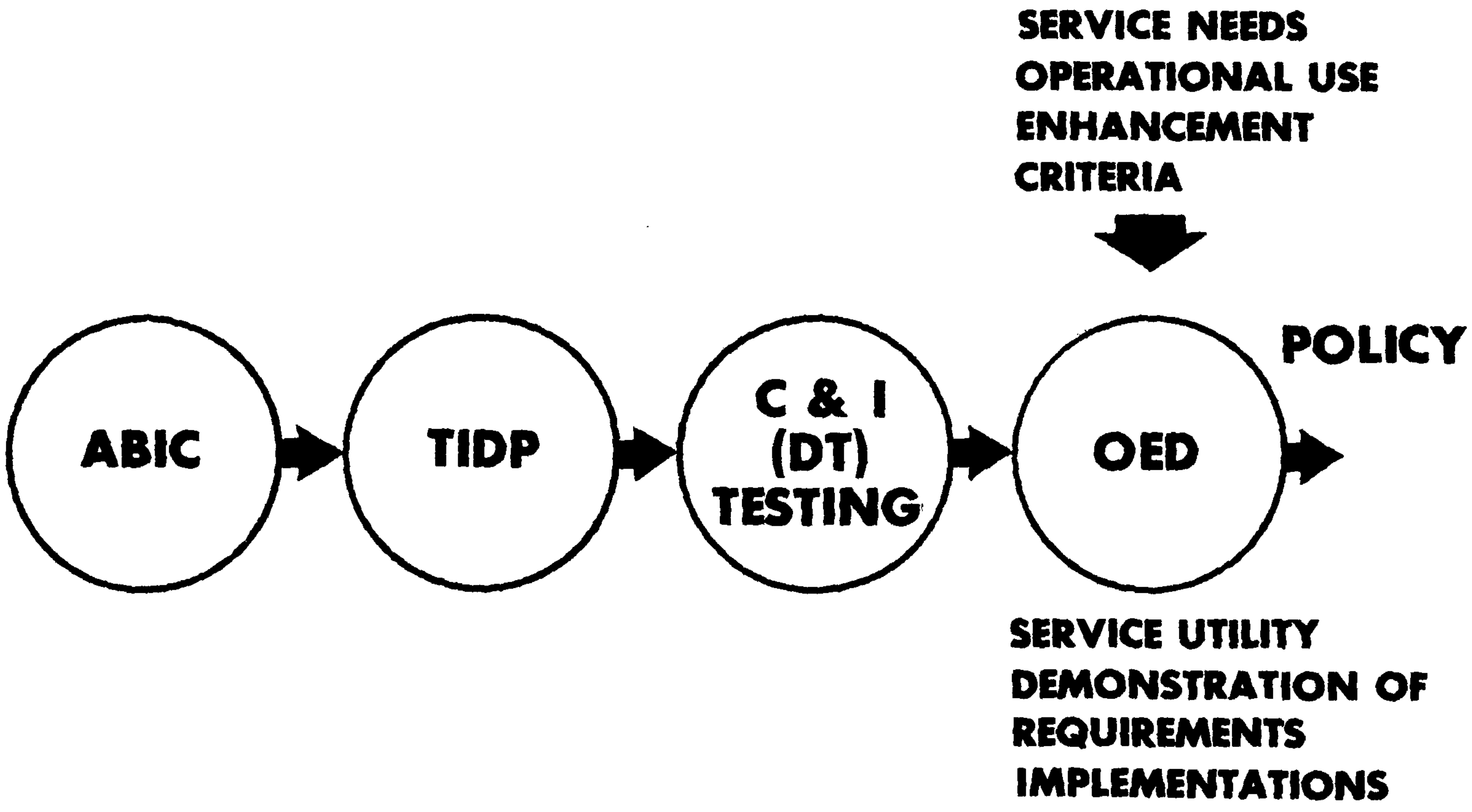
The reader may not agree with the challenges. If you are a R&D software developer and disagree with the preceeding challenges, you are encouraged to review your development programs to determine how interoperability will be achieved. Please do not be silent if you have solutions. **HELP**!!

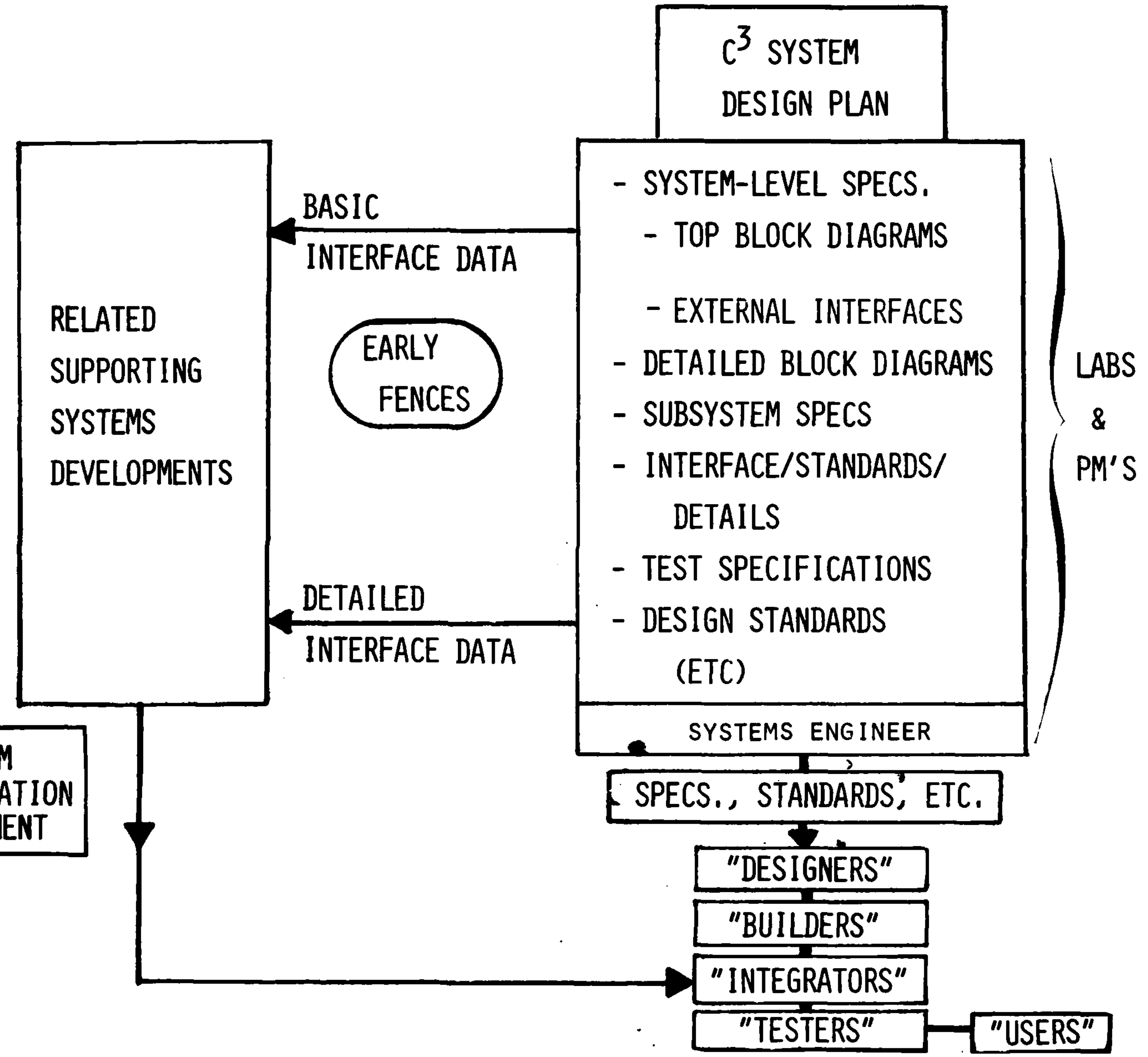

USERS (TRADOC)
OVERALL SYSTEM INTEGRATION GUIDES
CHALLENGES
REQUIREMENTS NEGOTIATION LOOP
REQUIREMENTS
ARCHITECT
- SYSTEM CONCEPTS
- ANALYTICAL STUDIES
- REQUIREMENTS ANALYSIS
- TRIAL CONFIGURATIONS
- TRADEOFF STUDIES
- FUNCTION/COST, RISK
INTEROPS III A&B
INTACS
C^3 SYSTEM DESIGN PLAN

CONCEPTUAL RELATIONSHIP OF INTEROPERABILITY PROGRAMS



INTEROPERABILITY METHODOLOGY





C3 SYSTEM DESIGN PLAN

- SYSTEM-LEVEL SPECS.
  - TOP BLOCK DIAGRAMS
  - EXTERNAL INTERFACES
- DETAILED BLOCK DIAGRAMS
- SUBSYSTEM SPECS
- INTERFACE/STANDARDS/ DETAILS
- TEST SPECIFICATIONS
- DESIGN STANDARDS (ETC)

SYSTEMS ENGINEER

LABS & PM'S

BASIC INTERFACE DATA

EARLY FENCES

DETAILED INTERFACE DATA

RELATED SUPPORTING SYSTEMS DEVELOPMENTS

SPECS., STANDARDS, ETC.

"DESIGNERS"

"BUILDERS"

"INTEGRATORS"

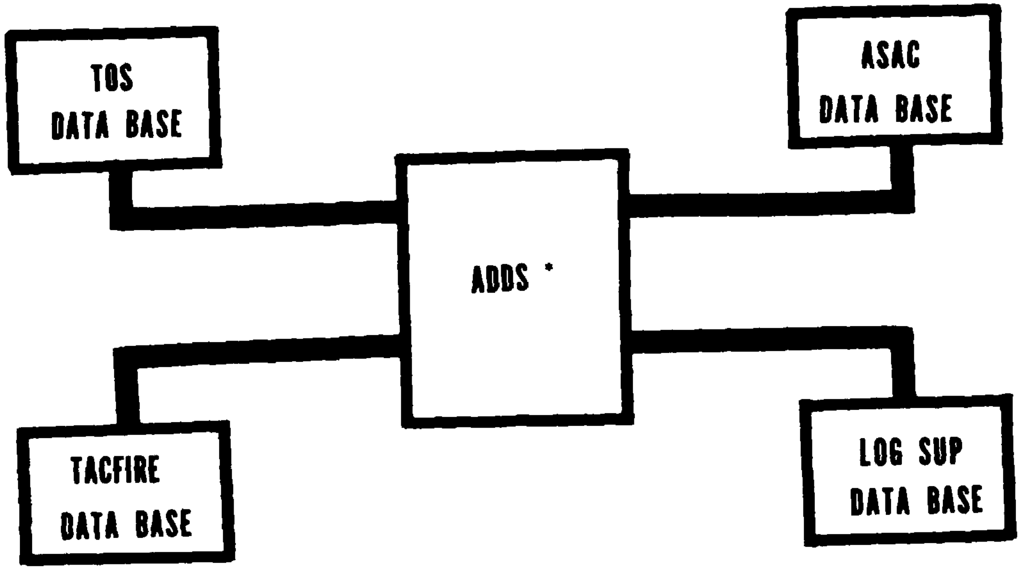
"TESTERS"

"USERS"

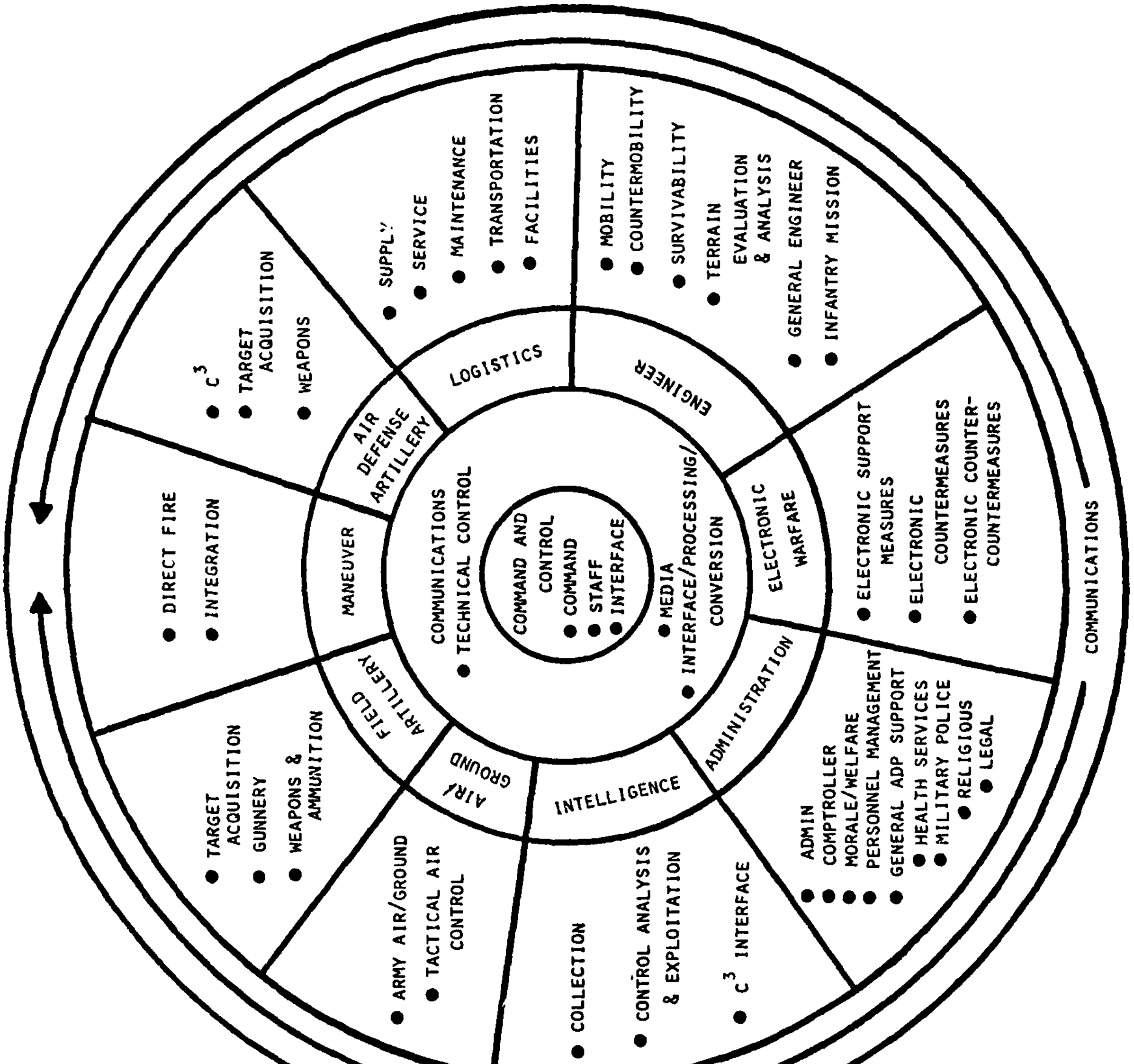
SYSTEM CONFIGURATION MANAGEMENT

CONOPS/SURVIVABILITY

TRANSFER OF DATA BASES



* ADDS, AR ARMY DISTRIBUTED DATA SYSTEM


COMMAND AND CONTROL
● COMMAND
● STAFF
● INTERFACE
COMMUNICATIONS
● TECHNICAL CONTROL
● MEDIA
● INTERFACE/PROCESSING/ CONVERSION
MANEUVER
● DIRECT FIRE
● INTEGRATION
AIR DEFENSE ARTILLERY
● C3
● TARGET ACQUISITION
● WEAPONS
LOGISTICS
● SUPPLY
● SERVICE
● MAINTENANCE
● TRANSPORTATION
● FACILITIES
ENGINEER
● MOBILITY
● COUNTERMOBILITY
● SURVIVABILITY
● TERRAIN EVALUATION & ANALYSIS
● GENERAL ENGINEER
● INFANTRY MISSION
ELECTRONIC WARFARE
● ELECTRONIC SUPPORT MEASURES
● ELECTRONIC COUNTERMEASURES
● ELECTRONIC COUNTER-COUNTERMEASURES
ADMINISTRATION
● ADMIN
● COMPTROLLER
● MORALE/WELFARE
● PERSONNEL MANAGEMENT
● GENERAL ADP SUPPORT
● HEALTH SERVICES
● MILITARY POLICE
● RELIGIOUS
● LEGAL
INTELLIGENCE
● COLLECTION
● CONTROL ANALYSIS & EXPLOITATION
● C3 INTERFACE
AIR/GROUND
● ARMY AIR/GROUND
● TACTICAL AIR CONTROL
FIELD ARTILLERY
● TARGET ACQUISITION
● GUNNERY
● WEAPONS & AMMUNITION
COMMUNICATIONS

MANAGEMENT CONTROL TECHNOLOGY

Allan Curry
AIRMICS

MANAGEMENT CONTROL TECHNOLOGY

SESSION CHAIRPERSON: Allan Curry

AIRMICS

SESSION SUMMARY

Throughout the software life cycle there is a growing realization that the causes of development failures cannot be narrowed to any one factor. The successful system development requires complete, consistent, and judicious application of managerial oversight.

Ms. Eldridge (CENTACS) discussed the type of planning necessary even after a system has been completed. One recent study indicates that over 50% of System Resource Requirements are expended after system implementation. Post-deployment effort should not necessarily be classified as maintenance. Dr. Donovan Young (AIRMICS) discussed a proposed decision aiding system for Software Development Management. This system will aid in the planning phase, supporting each step (from actual or what-if-requirements, through PERT analysis, to a proposed schedule, to life-cycle curves) with interactive provision for backtracking, testing tentative changes, and manipulating graphical representations of PERT networks, bar charts, and life-cycle curves. The final speaker, Mr. Herman (Manager of Systems Analysis and Development, Shell Oil Company), discussed implementation of management controls during the Software Development Life Cycle in a non-governmental management environment.

Post-Deployment Software Support for Army Defense Systems

Ingrid A. Eldridge

Systems Validation Division

CENTACS

This paper described the effort that is going on to develop an overall plan for post-deployment software support for Army Defense Systems. These systems are in various stages in the acquisition cycle. Presently, the Army has no overall, coordinated, well-defined plan of this type. There is little coordination taking place in this area and plans range from none-at-all to well-developed plans for a particular system. The Army can no longer afford a fragmented approach to software support over the life cycle of its systems. There is a projected demand for resources that probably cannot be supported.

This plan is being developed to rectify that situation. The goals of the plan were discussed. In addition, coordination is required with users, schools, developers, and personnel from other programs that impact this effort. Both technical and administrative management is involved and this was described. Finally, a description of the accomplishments to date has been given.

POST DEPLOYMENT SOFTWARE SUPPORT FOR ARMY DEFENSE SYSTEMS

Ingrid A. Eldridge

US Army Communications Research
and Development Command
Center for Tactical Computer Systems
Systems Validation Division
Fort Monmouth, New Jersey

SUMMARY

This paper describes a management problem that the Army has in devising an overall plan for the post deployment software support of its defense systems. A description of how this problem is being approached, as well as the status to date, will be given.

INTRODUCTION

The Army has over 170 defense systems in various stages of acquisition. Those systems which have post deployment software support plans call for yearly expenditures of 1 - 20 million dollars per system. When you multiply this by the number of systems involved, it is unlikely that the Army can continue to provide the projected demand for resources called for by separate software support centers. This is not only a funding problem, but also a problem in acquiring enough skilled personnel to provide the required software support.

To give an idea of the dimensions of the problem, software now accounts for almost ninety percent of the costs of a software system. In 1977, the Federal Government spent 4 billion dollars for software. Seventy-five percent of this went for software maintenance. Much of this went into correcting programming errors that should not have been there in the first place.

Many of the computer programs for the Army systems contain thousands of lines of code. It is usually not possible nor practical to precheck all the logic paths and program input values. Presently, the Army does not have many systems containing embedded computers in the field, but this is rapidly changing. In addition, as the user sees the "nice things" computers can do, he wants to take more advantage of them. This results in larger and more complicated programs. Large programs also result from making the system easier for the soldiers to operate.

Presently, the Army has a fragmented approach with no central coordination or control for the post deployment software support for these systems. To correct this situation, the US Army Materiel Development and Readiness Command (DARCOM) has tasked the Communications Research and Development Command (CORADCOM) to develop an overall plan for post deployment software support for Army defense systems. We began work on this effort in July. This paper will describe the problem, our plan of action, the requirements for PDSS, and what has been accomplished to date.

BACKGROUND

The Army systems that we are considering in our post deployment software support plan have been developed independently. Little effort was made to standardize hardware or software.

There are many different computers. Some systems have more than one type of computer in that system alone. Peripherals are of different types, and even if they are performing similar functions, if they are made by different manufacturers, they are usually different enough so that the soldier will require some retraining in going from one system to another. Computer languages are also different and vary from higher order languages to assembly and machine languages and microcode. Operating procedures vary; so do procedures for reporting software problems. Different forms are used. They are submitted through different routes and require different procedures. Application programs also vary. This is true, not only because there are different types of applications, but, if the same program was developed by a different contractor, it would probably be different. Although there are too many differences, there are similarities. Part of our jcb is to identify these. With proper planning, the number of similarities should increase in the future.

We would not have many of the software support problems we are facing today if software had been developed with a total systems approach.

Another problem is that, when software is delivered, it is often not properly documented. It is, therefore, impossible to maintain the software without contractor support. Even contractors have their problems. There are systems today that were not properly documented, and, since the employee left the contractor, present procedures are to patch around that section of code, since no others at the contractor's plant can understand it. Software test sets often are not adequately provided when programs are delivered.

A contributing factor to the large amount of post deployment software support required results from the system requirements not being adequately defined initially. It is often not until a system is operational that the user realizes how he really would like the system to operate.

Post deployment software support is not given the emphasis it should have when the system is being developed. If more resources were applied to this area early in a systems development, the software support would be better and less costly in the long run.

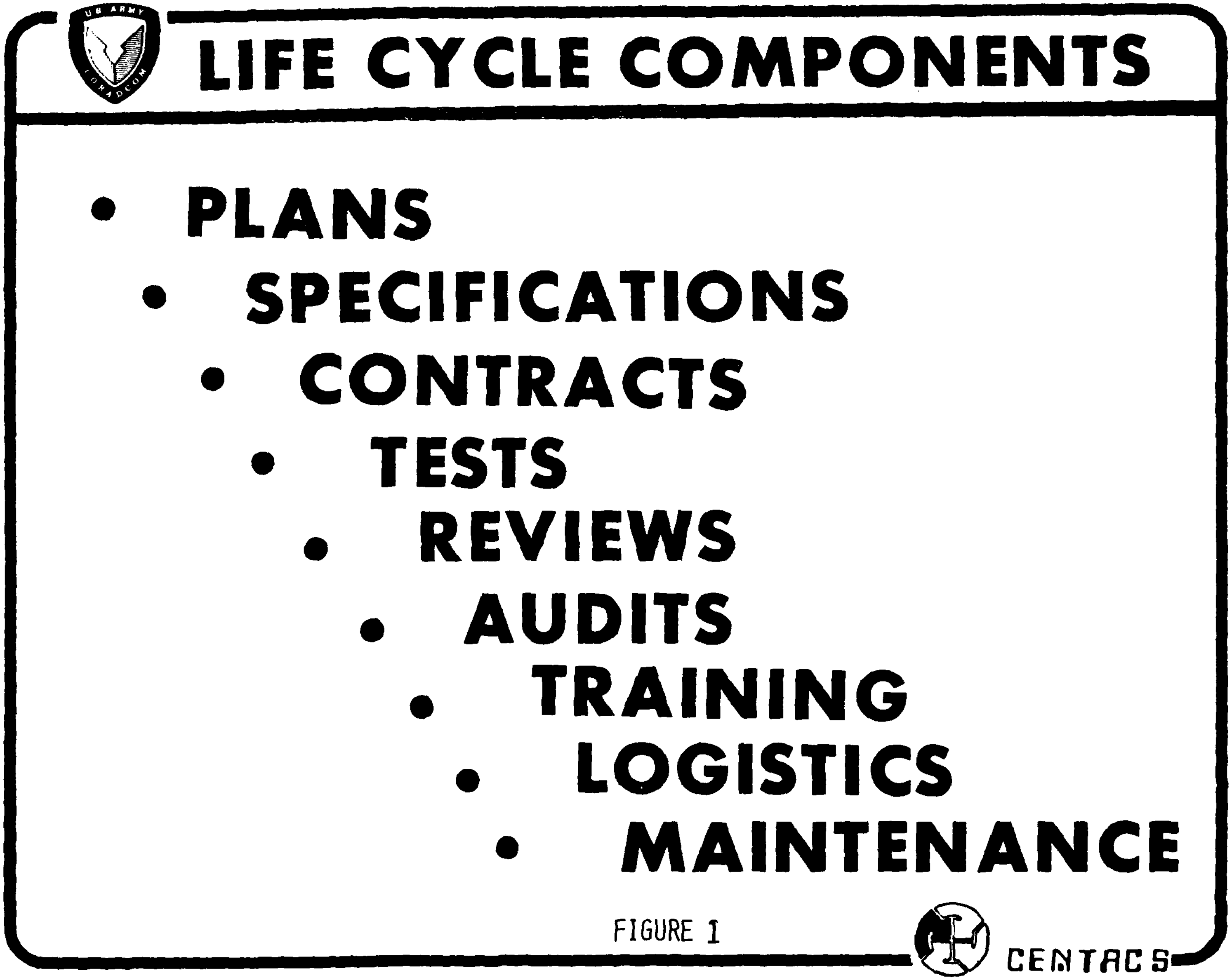
The Army, at present, does not provide for an organization to develop and administer PDSS policy on an Army-wide basis. Guidance is not adequate in this area. The funding situation is confused. This is due to a lack of central control.

There are regulations and standards addressing post deployment software support. They exist at all levels. For example, the Department of Defense has them; the Army, in turn, issues its own, and then the individual commands have their interpretations. The myriad of regulations and standards are confusing to a system developer who only occasionally has to consider developing a PDSS plan.

Although the guidance exists, it is often not followed. There are many reasons. One is that there are too many documents to refer to. They are not complete in themselves, so a developer has to refer to many to get a complete understanding. This is time consuming and difficult. There is redundancy from one document or another. Sometimes opposite instructions can be interpreted by going from one document to another. There is no step-by-step procedure to be followed. For the non-computer type, it is difficult to follow this guidance. There is no stress, enforcement, or incentive given to insure each system has a PDSS plan and that it is implemented and adhered to. Usually the developer is so busy developing his system and getting it to work that he has no time for PDSS. This is wrong. Time and resources should be allocated for PDSS from the very beginning. There have to be workable methods of enforcement and incentives to insure post deployment software support is provided for early enough.

See Figure 1 for the many other documents and factors that affect post deployment software support.

There are many problems with the procedures that are in existence for PDSS. One of the underlying causes is that procedures are developed independently, and there is little standardization from one system to another. We are not learning from past mistakes. It takes too long to get software changes to the field. Procedures for paying for PDSS are not standard.



LIFE CYCLE COMPONENTS

- PLANS
- SPECIFICATIONS
- CONTRACTS
- TESTS
- REVIEWS
- AUDITS
- TRAINING
- LOGISTICS
- MAINTENANCE

FIGURE 1

CENTACS

The resources used to provide PDSS have many problems that have to be resolved. There are personnel problems. One of the complaints is that military computer personnel are trained in COBOL and business applications rather than languages (such as TACPOL) that are required for Army defense systems. In addition, there is too frequent a turnover in personnel. Planning for these resources is too late. There is a lack of standard PDSS tools. There is too much duplication of resources since each system has its own software support center. No overall categorization of criticality of support has been made.

Reasons for changes to software, besides correcting problems that should not have been there in the first place, occur when a system's performance is increased. This could be implemented by changes in software or hardware. The latter could impact the software and require changes there also. Interoperability is a complex issue that often results in software changes. The Army has requirements that many of its systems interoperate. For example, the Tactical Fire Direction System (TACFIRE), a computerized artillery system, must interoperate with the Tactical Operations System (TOS) which keeps track of the enemy and friendly situations.

The Army also has requirements to interoperate many of its systems with those of the other services and agencies, such as for the Joint Interoperability for Command and Control Systems program (JINTACCS). International requirements present a further complication.

Changes in military doctrine must also be implemented in software. New hardware may also impact software for the same system or an interoperating one.

PLAN OF ACTION

In order to develop this plan, we have formed a task force (see Fig. 2) consisting of members of the various Army commands, including user representatives, developers, software support centers and schools. Some of the participants to date are shown. We require the benefit of a wide range of experiences in order to put together a meaningful plan. Task forces have been set up to address various issues and provide draft information for the group to review.

We are operating in the following manner (Fig. 3) to develop our plan. The schools, users, and developers provide the benefit of their experiences to the working groups who draft portions of the plan for the task force to review. It will then be reviewed by the schools, users, and developers before submission to DARCOM and subsequently DA.

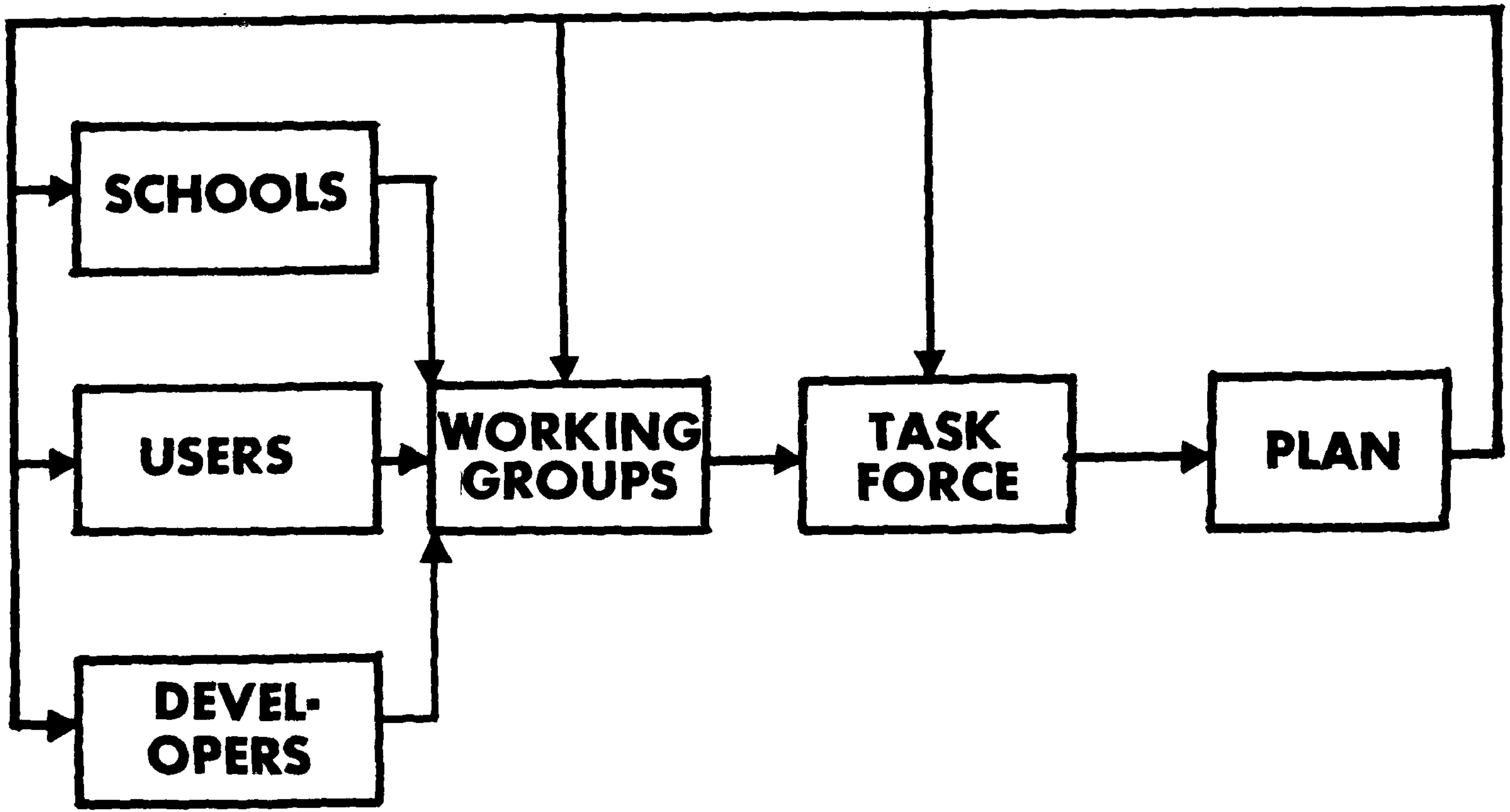
COORDINATION

- HQ, DARCOM
- TRADOC
- CACDA
- SCHOOLS
- PROJECT MANAGERS
- DARCOM R&D AND MATERIEL READINESS COMMANDS
- ARMY COMMUNICATIONS COMMAND (ACC)
- COMPUTER SYSTEMS COMMAND (CSC)
- ARMY AUTOMATION DIRECTORATE, HQ DA
- ARMY FIELD ARTILLERY BOARD
- ARMY MATERIEL SYSTEMS ANALYSIS ACTIVITY (AMSAA)
- OTHER SERVICES (JINTACCS)
- OTHER PROGRAMS (E.G. MCF)

CENTACS

FIGURE 2

Post Deployment Software Support Organizational Structure



SCHOOLS

USERS

DEVEL-
OPERS

WORKING
GROUPS

TASK
FORCE

PLAN

FIGURE 3

CENTACS

In developing the plan, we always keep in mind who our user is. Our goal is to provide the very best software support to the soldier in the field while working within time and dollar restrictions. Ours is not the first effort in planning Post Deployment Software Support, so we are reviewing the work that has gone on before for the Army. We are also looking at the experiences of industry and other military services.

We have developed a plan of action which we call our analysis plan. Our approach in performing this analysis is to use the successful experiences of others and avoid remaking the same mistakes. We believe this will allow us to develop a PDSS plan for short, mid- and long-term implementation. This analysis task force must look at the overall picture to provide a workable management plan with a means for enforcement.

As a result of this analysis we intend to recommend a structure for PDSS which includes procedures, guidelines, and standards. These will be provided as part of an overall report whose outline is shown in Fig. 4. We are preparing this report by forming working groups to contribute to the various sections. The problem and the facts bearing on it have been defined by the working groups. We are using the procedure of coming up with several alternative plans for post deployment software support, determining a strategy for analyzing the alternatives and then developing the plan that is selected.

Next I will discuss our schedule for developing the PDSS plan as shown in Fig. 5. We have attended meetings that were held at the Signal Center, Fort Gordon, Georgia, where TRADOC or the Training and Doctrine Command requirements were determined and given to us. We have had two working sessions at Fort Monmouth and plan a series of these as well as reviews by the Army management. Our draft should be completed by February and sent out for review with the plan submitted to DARCOM by the end of April 1979.

ASSUMPTIONS

Some of the assumptions we are making in developing this plan are as follows. The first is that it is mandatory that every Army defense system containing computational resources have a post deployment software support plan that is prepared early in a system's development.

Another assumption that has led to quite a bit of discussion is that we are considering firmware as a subset of software. It does not matter whether software is stored on a chip or another media such as tape or disk; it is still software.

The user will be an independent tester. The systems we are considering are not only the application systems themselves, but any software support

Post Deployment Software Support

Product

SECTION I	● INTRODUCTION›	METHODOLOGY, ASSUMPTIONS
SECTION II	● PROBLEM DEFINITION›	REQUIREMENTS
SECTION III	● BACKGROUND›	FACTS
SECTION IV	● ALTERNATIVES›	OPTIONS

FIGURE 4A

CENTACS

Post Deployment Software Support

Product

SECTION V • ANALYSIS > ALTERNATIVES { ADVANTAGES / DISADVANTAGES

SECTION VI • RECOMMENDED PLAN > DETAILED DEVELOPMENT

SECTION VIII • SUMMARY > CONCLUSIONS, RECOMMENDATIONS

FIGURE 4B

CENTACS

US ARMY CORADCOM

Post Deployment Software Support

Schedule

TRADOC REQUIREMENTS MEETING I	6-7 JULY 1978
TRADOC REQUIREMENTS MEETING II	8-11 AUGUST 1978
ORGANIZATIONAL AND WORKING SESSION I	23-25 AUGUST 1978
WORKING SESSION II	3 OCTOBER 1978
WORKING SESSION III	2 NOVEMBER 1978
GENERAL OFFICER REVIEW	5 DECEMBER 1978
WORKING SESSION IV	12-13 DECEMBER 1978
COMMAND/STAFF REVIEW	9 JANUARY 1979
WORKING SESSION V	23 JANUARY 1979

FIGURE 5A

CENTACS

Post Deployment Software Support

Schedule

DRAFT PUBLISHED FOR REVIEW → 1 FEBRUARY 1979

REVIEW COMMENTS RETURNED → 5 MARCH 1979

REPORT SUBMITTED TO DARCOM/DA → 30 APRIL 1979

FIGURE 5B

CENTACS

equipment. Some of this equipment could be commercial automatic data processing equipment. The support equipment could be found in test beds or training facilities as well as software support centers. The software involved is for the system peripherals as well as the computers.

Other assumptions are that PDSS is required for changes due to correcting problems, adding enhancements, and to meet new requirements.

In addition, it is assumed that it will be the responsibility of the user to report and document software problems. Support will be for systems sold to foreign countries and for systems the Army is using but did not develop.

PDSS REQUIREMENTS

The following are a list of the requirements the task force has specified for PDSS (see Fig. 6). We took the list submitted by TRADOC (which will be given subsequently) and added others to it. We have to consider how software changes affect interoperability. Testing is required after any software change. This is a difficult issue. Presently, it sometimes takes as long as 18 months to get a software change into the field because of requirements to go back through TECOM and OTEA testing even though the change is minor. We are working to come up with a solution to provide a better response to the field in this area. Standards have to be adhered to. Configuration management must be performed. This is a significant undertaking where many systems are involved, and they are located worldwide.

Systems that our Government has sold to foreign countries must also be supported. A separate working group is addressing this issue. This is also true of the systems that the Army uses but did not develop. These include systems developed by other services and now, even systems developed in other countries. Resources include personnel and funding.

Requirements given to us by TRADOC are shown in Fig. 7. The first is that all systems must have an organization or organizations responsible for their PDSS. The software support must meet the requirements of the user. The plan for software support should be jointly developed. Also, new equipment is not to be issued if it affects software until it can be accompanied by a software update. Furthermore, TRADOC sees no requirement for type classifying the software. (See Fig. 8) - The software support organization must keep the documentation current as software changes are made. Any changes made to software should be done while maintaining the operational capability specified in the requirements documents. PDSS is required on the battlefield - worldwide. Software includes that required to accommodate changes in training. Another TRADOC requirement is that configuration management must be performed according to a management plan.

PDSS REQUIREMENTS

- ● SUBMITTED BY TRADOC
- ● INTEROPERABILITY
- ● TESTING
- ● STANDARDS
- ● CONFIGURATION MANAGEMENT
- ● FOREIGN SUPPORT
- ● SYSTEMS NOT DEVELOPED
- ● RESOURCES

CENTACS

FIGURE 6

TRADOC REQUIREMENTS FOR PDSS

- ALL SYSTEMS
- RESPONSIVE TO USER
- CLOSE COORDINATION - SUPPORT ORGANIZATION AND USER
- NEW EQUIPMENT AFFECTING SOFTWARE - ACCOMPANIED SOFTWARE UPDATE
- NO TYPE CLASSIFICATION OF BATTLEFIELD AUTOMATED SYSTEM SOFTWARE

FIGURE 7

CENTACS

TRADOC REQUIREMENTS FOR PDSS ORGANIZATION

- MAINTAIN DOCUMENTATION
- SUSTAIN SYSTEM OPERATION
- COMBAT RESPONSE - WORLDWIDE
- TRAINING SUPPORT SOFTWARE
- CONFIGURATION MANAGEMENT VIA PLAN

FIGURE 8

CENTACS

TRADOC has also given us the following requirements for the user representative. One is that it is up to each user representative to specify the time it requires for a software change to be implemented. Another is that TRADOC must approve changes affecting the "what" of the system, but it is up to the PDSS organization to determine how these changes are implemented. Finally, the user representative must be a voting member on the configuration control board. If a software configuration control board is set up, the user representative must also be a voting member on that board.

The issue of wartime and crisis support is a difficult one. However, we all must remember that there is a tradeoff in cost and responsiveness

PDSS FUNCTIONS

Figure 9 lists software support functions. One is that as soon as a hardware change is recommended, the impact on software be evaluated. This includes how long it would take to implement the software change and what it would cost. The actual implementation of the software change is an obvious function. Distributing and approving these changes is a large undertaking. Test programs have to be developed.

Software models are also a PDSS function. Where test bed facilities are used, PDSS includes the development of simulations and emulations. This could be especially true of systems requiring interoperability testing. Testing and retesting is another function. Software standards must be maintained. Various types of field support must be provided to the user. This is another area where there is much disagreement. Some types of systems, such as intelligence, desire software changes on the battlefield to accommodate unknown changes in the threat. Testing includes interface testing where this is a requirement.

Considerations in developing the PDSS plan are many. They include: location, personnel, security, equipment, documentation, training, and distribution of updates. Personnel with both technical and management as well as various specialized abilities are required. The personnel required to perform PDSS are mostly highly skilled. These include: analysts and/or engineers, programmers, computer operators, key punchers, administrative, library, test, user representatives, documentation, logistics, and instructors. Shortages of these personnel have been predicted for the future.

POTENTIAL SHORT TERM ACTIONS

Since we are bringing together the experiences of many involved in PDSS, we decided that there could be some recommendation of early improvements that could be made without waiting for the development of the overall plan.

For example, an effort could be made to standardize as much as possible the procedures the user must follow to report a software problem. This would include standardizing the forms to report software problems. We have found terms for the same thing vary from system to system. For example, an application program is known by some as a parametric program. Differences in nomenclature lead to unnecessary problems for all concerned with PDSS. Enforcing the development of a PDSS plan is another action that could be taken early. So is the centralization of control. Impact on software should be studied for the time it takes as well as the cost involved. This should be done as soon as a hardware change is proposed. The funding should be set up so that there is no problem for all users to obtain updated versions of the software and accompanying documentation. We understand this is a problem in the Field Artillery Digital Automatic Computer (FADAC) system.

Some of the long-term actions that the plan will provide are an in^reased sharing of resources. In addition, when new systems are developed, every effort should be made to use existing programs and hardware, if at all possible. Effort should be made to see that all software support centers and training facilities are using the latest techniques. Finally, central coordination of the PDSS program is required for all Army defense systems.

SUMMARY

We believe an overall plan for PDSS will reduce the number of daily crises that occur. A plan that shares resources will provide many advantages. (See Fig. 10.) Some are obvious; others are indirect. In summary, we believe that a plan that is developed with the assistance of everyone involved in PD3S can provide the Army with the best use of its resources for post deployment software support.

POST DEPLOYMENT
SOFTWARE SUPPORT FUNCTIONS

- HARDWARE CHANGES
- DISTRIBUTE AND APPROVE SOFTWARE CHANGES
- DEVELOP BENCHMARK AND SCENARIO TEST PROGRAMS
- DEVELOP MODELS
- DEVELOP SYSTEM SIMULATIONS AND EMULATIONS IF REQUIRED
- PERFORM TESTING AND ANALYZE RESULTS
- MAINTAIN SOFTWARE STANDARDS
- PROVIDE FIELD SUPPORT TO USER
- PERFORM INTERFACE TESTING

FIGURE 9

CENTACS

ADVANTAGES OF POOLED SUPPORT

REDUCES

- PERSONNEL AND EQUIPMENT
- TECHNICAL BASE
- "BOOKKEEPING"
- LOGISTIC PROBLEMS
- TRAINING
- DUPLICATION OF EFFORT

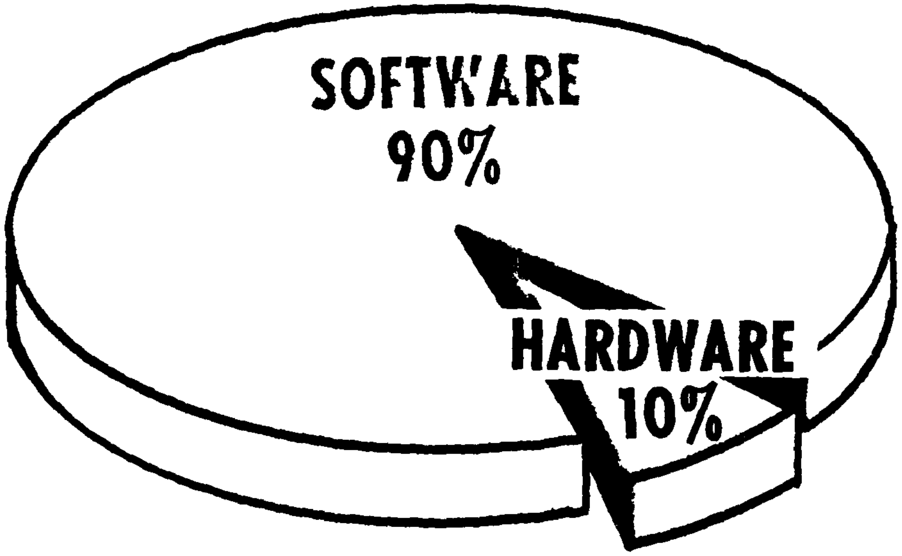
IMPROVES

- CONTROL AND CONVERGENCE
- FUNDING REQUIREMENTS
- TECHNICAL SYSTEMS
- STANDARDIZATION

FIGURE 10

CENTACS

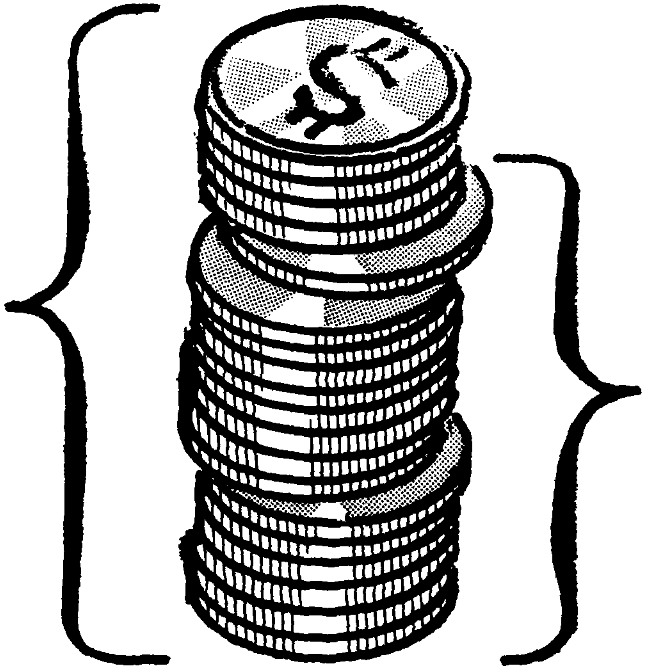
COSTS OF DEVELOPING A COMPUTER SYSTEM



CENTACS

US GOVERNMENT TOTAL SOFTWARE COSTS

1977



$4 BILLION

75% MAINTENANCE

CENTACS

A Decision-Aiding System for Software Development Management

Donovan Young

AIRMICS

AIRMICS is developing a prototype decision-aiding system for software development management, intended for implementation on a small computer system with some graphics capability. Software-breadboard development is being done using a large-scale computer. The system is being designed according to Decision Support System (DSS) specifications to support semi-structured interactive use by non-programmers. The initial modules include an updateable data base, a "what-if" or scratch data module, an interactive PERT module, and an interactive life-cycle cost profile module. Provision is made for incorporating additional automated tools for software life-cycle management as they become available.

A DECISION-AIDING SYSTEM FOR SOFTWARE DEVELOPMENT MANAGEMENT

Donovan Young
AIRMICS
Atlanta GA

AIRMICS is in the early stages of developing a prototype decision-aiding system for helping software project managers accomplish certain planning tasks: PERT analysis, time-cost tradeoff analysis, resource allocation, and generation of activity schedules and life-cycle curves. The initial simulated prototype design carries the name Software Project Management Planning System (SPMPS), and is conceived to be part of a Software Project Management System that will eventually include real-time planning tools, visibility tools, and perhaps control tools and management communication tools. The system is being designed according to decision support system (DSS) specifications to support semi-structured experimental-mode use by non-programmers. Software-breadboard development is being done using a large-scale computer system.

Background

Findings and recommendations of the Second Software Life Cycle Management Workshop (Atlanta GA, 21-22 August 1978) emphasized the need for real-time automated management tools for software managers. Also indicated was a need to investigate the use of PERT analysis as an alternative to curve-fitting in generating life cycle cost and resource-utilization curves. SPMPS addresses these needs directly.

Current research in real-time automated management decision aids centers around the idealized concept of the Decision Support System (DSS), which is a system to aid a non-programming individual to solve semi-structured ad-hoc decision problems experimentally by interactive access and manipulation of stored data, "what-if" data, and software transformational modules. DSS researchers agree that the crucial problem for this kind of use is the problem of interaction with the user. Ad-hoc decision problems, being one-time problems, do not call for production-code efficiency from the data processing standpoint; in fact, the emphasis is entirely at the other end of the spectrum. Every feature of a DSS is designed for user convenience. High computer overhead is expected and condoned.

DSS implementations are normally characterized by advanced graphics, calculator-mode operation, basic statistical manipulation, ad-hoc report generation, and provision of special-purpose modules designed for the set of jobs at hand. Experimental use is emphasized; the user is expected to come to the DSS terminal for help in modeling a problem, not simply to solve it after off-line modeling. DSS researchers distinguish between 'interactive' and 'conversational' modes of interaction; in conversational modes the computer is in control and issues prompts, whereas in interactive modes the user is in control. DSS designers concentrate on truly interactive modes where possible.

It is generally considered good practice in DSS design to let the user/machine interaction emulate the communications that the user would undertake while solving the same problem unaided. Although little real research has been reported along these lines, it has been repeatedly suggested that DSS designers should objectively observe the user in his decision-making environment, recording his questions and commands as he interacts with staff, resource persons, and subordinates--then let the user's most-used imperative verbs be incorporated as commands in the DSS and let the user's most-used nouns be incorporated as data elements.

Communication goals of DSS, then, are to emulate the decision maker's established modes of communication as closely as possible. Since ordinary (aidless) decision making is usually a seeking, probing, experimental affair (as contrasted to monolithic acts such as modeling and solving the problem as an operations research optimization problem), the preferred mode of DSS interaction is experimental. Designers try to encourage experimental use by providing interactive documentation and diagnostics, and even by providing a full enough spectrum of alternative commands (and a transparent enough logical structure) that the user can profitably enter a command he does not know in the hope it will have the desired effect.

The Software Project Management System

The Software Project Management System ultimately to be fielded is envisioned as a DSS implemented on a stand-alone microprocessor or minicomputer system to provide planning aids, visibility aids, control aids, and general aids to the software project manager. The visibility modules of the DSS would receive progress input (data on actual expenditures, actual milestone completions, etc.) from the MIS, and would incorporate forecasting software and 'variance analysis' (in the cost-accounting sense) software; it would output 'visibility' reports such as bar charts showing actual and planned progress side by side, with forecasted progress shown in contrasting graphics. The control aids would include algorithms for automated flagging of exceptions, and perhaps automatic preparation of progress reports up or down the chain of command. The general aid modules would include ad-hoc report generation (histograms, cartesian plots, pie charts, etc.), basic statistical computations, and calculator-mode capabilities along with usual desktop computer capabilities.

SPMPS will include only planning aids.

The Initial Simulated Prototype SPMPS

SPMPS is planned to contain a limited number of interactive planning tools, namely

...interactive PERT analysis

...interactive time-cost tradeoff and resource allocation analysis

...generation of schedules and life-cycle curves.

These tools, by themselves, constitute a useful management-aid package for a software project manager. The basic planning methodologies are well known, and they are well adapted to interactive use. The time-cost tradeoff and resource allocation procedures, particularly, are heuristics that are easier to perform with a user/computer team than with a computer alone.

A preliminary list of the initial capabilities is as follows:

1. Interactive development of task breakdown of a project into activities, with specification of activity durations using the PERT-beta method or the newer Schick & Lin interactive probability estimation routine, and specification of resource utilizations.

2. Respecification (temporary or permanent, cumulative or non-cumulative) of activity parameters at any point in the session.

3. Representation of the current list of activities as a standard activity-on-arc PERT graph.

4. Solution of the PERT graph, with reporting of early start times, slack times, etc.

5. Representation of an early-start-time schedule as a Gantt chart.

6. Interactive 'crashing' and 'sliding' of activities.

7. Interactive reporting of resource utilizations.

8. Interactive generation of life-cycle curves.

Work Plan

The initial phase, to design, implement and demonstrate a simulated prototype of SPMPS, includes the following tasks:

1. Identify a proponent within the Army who has large-scale software project management responsibility and who is willing to provide initial consultation and data, and who also will indicate willingness to corsider allowing testing of the simulation prototype SPMPS under appropriate conditions in an actual software project development environment.

2. Design the Software Project Management Planning System simulated prototype. This includes implementing the management-aid modules (PERT planning, time-cost tradeoffs, resource leveling, scheduling and life cycle curve generation), providing a design of the user/machine interface, and testing the simulated system under laboratory conditions.

3. Perform a limited experiment in the proponent's environment, including performing PERT analysis of a designated software project, along with resource leveling and scheduling, and compare the results with those of current planning techniques.

4. Analyze the limited experiment to obtain a preliminary evaluation indicating (a) whether the SPMPS concept is viable as a management tool, (b) whether use of a full-scale SPMPS would have been useful in the actual software project on which the prototype was tested, (c) what changes in the modules are indicated by the results of the experiment, and (d) what modules should be added and/or modified in follow-on work.

5. Propose the follow-on development work. This will probably involve design, implementation and testing of two or more configurations and the testing of additional modules, ending in a prototype ready for full testing.

Parallel Projects

Two other projects are planned in order to obtain parallel data and experience on special-purpose modules representing DSS components that typify those of the eventual Software Project Management System but whose design-relevant characteristics are not included in SPMPS. In both of these additional projects the needed core software already exists, and work can concentrate primarily on the user/machine interaction.

The first of these is a generalized decision tree module. (A generalized decision tree is simply a decision tree in which the activity durations are allowed to be independent random variables.) This typifies operations research modules that are extremely general in application, applying to many different kinds of problems. This module is similar to the modules around which some existing DSS's have been built.

The second of the two additional projects is advanced-graphics implementation of a graphical solution of the arbitrarily-constrained minimax location problem, which has recently been published and demonstrated by Rick Rosenthall of the University of Tennessee at Knoxville. This algorithm is new, it requires user participation for its solution, and it solves a narrow range of problems that are immediately understandable in a variety of military contexts. The module will typify those that will normally be used by a given decision maker in only one way. In a computer context, for example, the minimax algorithm allows location of a computer van to minimize the distance to the farthest user served, with arbitrary constraints such as a requirement to locate on a particular road or perimeter, or outside a designated area.

SPMPS and these two parallel projects will provide a fairly comprehensive initial test of the DSS concept applied to the provision of real-time automated management tools in software project management.

A BUSINESS APPROACH TO MANAGEMENT AND CONTROL OF THE SYSTEMS DEVELOPMENT PROCESS

L. T. Herrmann
MANAGER, SYSTEMS ANALYSIS AND DEVELOPMENT
SHELL OIL COMPANY
HOUSTON, TEXAS

Abstract

The successful development of large, one-of-a-kind, complex software systems in a multifunctional environment demands a total systems approach and an appropriate management process suited to the particular organizational structure and environment which it seves.

This paper identifies a number of problems common to today's systems development environment and presents one approach to their solution through establishment of a full spectrum of management processes, management aids and controls, management skills, and standards and training. These elements are considered essential for the achievement of successful project management results.

Acknowledgement

The processes and elements described herein represent the product of several years of effort by many dedicated professionals within Shell's Information Systems Department. I would like to give credit and recognitition to those collectively as well as those currently devoted to improving the entire systems development process. Additionally, I would like to thank those on our staff who assisted in assembling this paper for presentation.

A BUSINESS APPROACH TO MANAGEMENT AND CONTROL OF THE SYSTEMS DEVELOPMENT PROCESS

L. T. Herrmann
MANAGER, SYSTEMS ANALYSIS AND DEVELOPMENT
SHELL OIL COMPANY
HOUSTON, TEXAS

I. INTRODUCTION

A. Systems Development in the Service Organization

Systems development organizations may exist within a corporation, serving only the specific needs of that corporation, or as an independent profit center selling their services in the general marketplace. In either case, with the exception of profit motivation, they seem to share a common charter:

> To develop efficient, cost-effective, and secure data processing systems on schedule, within budget, and consistent with user objectives and requirements.

The independent business has always recognized the risks of operating in the open marketplace. The laws of supply, demand, and open competition govern its success or failure. Even in a market where demand for data processing services significantly exceeds the available supply, a business which cannot meet all of the components of the charter mentioned above is destined to fail.

The corporate users of service organizations and the development organization heretofore reluctant or unwilling to accept the structure and discipline of proven project management concepts and practices are now demanding project management of the ongoing application equivalent to that applied to capital projects. The service organization is responding. There is increasing concern for improved cost-effectiveness, control, productivity measurement and accountability. Managers are searching for improved methods and tools to help us understand and administer the development process. We question our organizational alignments and processes, and we constantly search for improvements that will insure project successes.

This paper outlines current philosophies, strategies, methods and tools being employed by Shell Oil Company in managing the application development process.

I. INTRODUCTION (Cont'd)

B. Current Problems for Management

The increasing interest in the management and control of the systems development process is not accidental. Many conditions in the current business, economic, and social environments indicate the need for improvement:

- Data processing expense as a percent of revenue trended favorably downward for the period 1970-74, but for the last three years has leveled at .7%. (See chart on page A-1.) The demands and pressures to continue the downward trend or hold it level with inflation require improved management to increase productivity.

- Personnel costs, as opposed to hardware/software costs, are taking an increasingly larger share of the data processing resource. There is an increased awareness and concern relative to the lack of systems development performance measurement criteria and historical data along with changing attitudes of professionals.

- Producing reliable software designed for change is far less expensive than fixing the system later on, and reducing maintenance costs allows reallocation of resources to new and profitable development. Current trends indicate the ratio of maintenance to development work is about 60 percent to 40 percent and rising.

- Systems are becoming more centralized, integrated, complex, and large in both size and function. They represent significantly higher investments to user organizations.

- Users are more mature and knowledgeable and are demanding that we function in the same business-like manner they expect in their own organizations.

- Data processing is more visible today because our installed systems have become an integral part of company operations and day-to-day business activity, rather than just being simple batch record keeping and reporting systems.

- Systems professionals including managers have been reluctant to accept discipline; they view systems development as more an art than a science. There is a tendency to rely heavily on experience subjectiveness, and intuitiveness. Such attitudes, if allowed to prevail, diminish the effectiveness of efforts to improve the planning and management processes and to achieve productivity benefits afforded by improved tools and methods available.

I. INTRODUCTION (Cont'd)

- Our performance record is being increasingly challenged by management; there is also increasing dissatisfaction on the part of the user with time and budget overruns, and with an unsatisfactory communication process.

- Technological, organizational and business changes occur rapidly and have a significant impact on the development process as well as our ability to respond.

- The user's view of "in-house" expenditure of discretionary funds has been changing over time. Discretionary spending is now being viewed on a more equal basis with outside capital expenditures. This view demands a more objective cost versus benefit analysis of any project undertaken, and increases demand for improved performance and productivity on the systems development organization.

C. Goals and Solutions

Understanding all of these factors may explain why we need an improved, systematic phased business approach to managing and controlling systems development, but it does not help us with the how to get there.

The development process most certainly can be improved to become more cost effective and productive. However, what is required is a total concept approach in a structured, well managed environment in which effective use of carefully selected productivity tools and a positive attitude of managers and professional staff prevails. (See chart on page A-2.)

The goals which we set for ourselves lie in the center of our target area, successful management and control of systems development. The outer rings of the target can be viewed as successive layers the Management Process, Management Aids and Controls, Management Skills, and Standards and Training needed to consistently reach the target area. Without such a total concept approach and supporting tools, hitting the bull's-eye (i.e., a successful project) is left to luck. The consistent bull's-eye is a result of a great deal of planning, management, skills development and much experience.

II. THE MANAGEMENT PROCESS FOR SYSTEMS DEVELOPMENT

An examination of the complete process for developing systems reveals there are certain key concepts which must be recognized and practiced to achieve successful management of a project. Clearly identified authority and accountability are absolute essentials. Early agreement on the statements of project scope, objectives, and requirements must be produced by initial project efforts as the foundation for all work to follow in the total project process. We must know the strategy and execution plans of the business to be served and obtain solid commitment for the support of our projects for these business goals. The following concepts have significant impact:

A. Key Concepts for Project Management

- Emphasize management of the process by establishing well-focused authority, responsibility and accountability. Clearly define and communicate roles, functional definitions, and sign-off authorities at the beginning of the development life cycle.

- Implement and adapt a systems management methodology to fit the management process and the organizational structure.

- Install a project management information system including elements such as staff skills requirements and availability, project control and tracking, change control, costing and billing, and project completion history. Use it to manage.

- Identify a sponsor in the user organization with sufficient authority and obtain a commitment. Insure proper communication processes are established and maintained throughout the project life.

- Establish a Systems Management Team within the data processing organization with authority to determine the degree of management required and to make timely decisions involving the balance of resources and commitments.

- Establish a project steering committee to guide matters of concern at the project level and make decisions within the project jurisdiction.

- Identify the appropriate Projects Manager and leader as the first line of management with responsibility through all project phases. Accountability for performance to plan is a requirement.

- Create a project team, including appropriate data processing and user representation in all phases, in a manner to obtain their commitment and assure their continuity for the project.

B. Goals of the Project Development Cycle

All projects are directed at common goals which are of prime interest to both user and data processing organizations. A completely successful project will achieve all of these goals, namely:

- Deliver efficient/effective systems
- Utilize appropriate technology
- Meet implementation schedules
- Achieve expected development and operating costs
- Satisfy user scope, objectives, and requirements
- Produce planned deliverables
- Provide a flexible system designed for future changes
- Minimize maintenance cost during the remainder of the operational life cycle
- Provide systems integrity and security

There are other goals to be achieved that are mostly germane to interests of the development organization. These are:

- Facilitate measurement of professional performance and productivity
- Assist in measurement of organizational performance and productivity
- Develop/maintain functional business knowledge and expertise
- Maintain/improve technical and professional skills
- Provide for optimum use of physical and human resources
- Improve and maintain credibility with the user

C. Organizational Functions for Project Management

The Information Systems Department (ISD) within Shell is organized in three internal departments according to broad areas of responsibility. These departments along with functional systems coordinators manage information systems for the corporation. Following is a brief explanation of the province and responsibility for each of these.

Systems Analysis and Development (SA&D) is responsible for all information systems development activities including systems analysis, systems design, programming and preparation of implementation plans. This paper refers to this department as the development organization.

II. THE MANAGEMENT PROCESS FOR SYSTEMS DEVELOPMENT (Cont'd)

Systems Support and Implementation (SS&I) is responsible for all control, maintenance, enhancement, and extension of existing information systems, user consultation for those systems, and the implementation function for new systems. This paper refers to this department as the support organization.

Systems Technology and Consultation (ST&C) is responsible for all professional training, techniques and standards, corporate data base management, and scarce skills utilization management. This encompasses techniques, standards, procedures, and advanced systems technology for business and technical sytems.

Systems Coordination is responsible for coordinating and assisting user organizations in the assigned functional area. This includes new systems requirements, funding, and coordination of all data processing activity carried on for the user.

In addition to the organizational functions described, there are additional components essential to the project management effort.

A user Steering Committee is responsible for a development project or group of related projects to direct the development team on policy, procedural and organizational matters affected by development activities. The committee monitors project activities to insure priorities, schedules, budgets, scope and objectives remain consistent with user desires. Most importantly, it makes key design and implementation decisions for the organizations affected by the development project.

The Quality Assurance function (discussed in detail on page 12 of this paper) serves as a managemet assurance that project plans, risk assessments, internal project reviews, and quality assurance audits are accomplished. In our organization, the quality assurance function reports to the development manager and also is responsible for areas discussed elsewhere, such as the project orientation phase, estimating review board, and project completion activities, along with all organizational performance tracking and reporting. In general, we consider the quality assurance a resource optimization function for our development organization in that plans or commitments and staff resource availability are in accord.

The Systems Management Team is a management strategy team consisting of managers of the development and support organizational elements discussed above. They function to determine degree of management required for development, implementation, and support activities and their primary function is to make decisions relative to balancing overall resources against commitments.

II. THE MANAGEMENT PROCESS FOR SYSTEMS DEVELOPMENT (Cont'd)

The team also serves as the authority to adjudicate any conflicts which may arise. Another key function is to combine develoment and support requirements and allocate resources based upon work plans.

User Management of each operating or service function in Shell has a focal point for systems development and support activities. The actual job function and title vary based upon size and complexity of the user requirements but typically would referred to as Manager Information Systems for a given function. In some cases a staff of liaison personnel supports such a function in addition to a high-level management team in the user organization with authority and responsibility for expenditures, work direction, priority setting, planning and budgeting, etc.

We also make extensive use of "user experts" or representatives as part of a resource matrix approach to development. During a certain phase of the development life cycle they may be assigned specific development activities or become part of a development project team. Another approach to maintaining functional user knowledge and involvement for system development is the assignment of user "operating specialists" to the development organization.

Project Coordinator - When a Projects Manager is responsible for a major system development effort comprised of a number of highly interrelated and interactive projects, a function is needed to provide consolidated visibility, planning and control. A separate position of Project Coordinator is identified to perform these functions normally shared by the Projects Manager and the Project Leaders. This position assists both the Projects Manager and the Project Leaders in the following ways:

- Provides the Projects Manager with additional time to perform required administrative duties and control functions.

- Allows the Projects Manager to be effective at a higher level of user interface.

- Allows the Project Leaders more time to concentrate on their specific projects, and enhances communication between those projects.

- Provides a focal point for direction of cross-project support services such as data base administration and project assistance.

The Project Coordiator function may require either a full-time assignment or part-time assignment, possibly shared with other project responsibilities.

General duties include:

- Responsibility for centralized coordination of functions common to all projects within a large systems development effort.

- Maintain central project visibility to users and project development staff.

- Coordinate project planning, design, and control.

- Status reporting and staff support activities between projects.

Normal administrative duties of the project manager/leader such as personnel, budget, direction of overall project activities, etc., are not included in this function.

Refer to charts for systems and project management on pages A-3 and A-4.

D. Management of the Project Development Cycle

The end products of the development activity are typically characterized as large, complex, one-of-a-kind systems, with the construction process being bounded by a difficult communication process and lacking a common blueprint mechanism between users and developers.

While the activity can be viewed as consisting of several discrete successive phases, the system development process in the real world rarely has clear-cut phase boundaries. The approach toward project management must recognize this and provide an atmosphere for managing variances in the process as a normal course of business.

A system development life cycle can be described by an orderly series of logically interrelated activities - groupings of which are referred to as phases. Phases are identified by discrete achievement points which serve as management decision opportunities and review points for past and future plans, accomplishments, and expenditures.

II. THE MANAGEMENT PROCESS FOR SYSTEMS DEVELOPMENT (Cont'd)

The system life cycle in Shell is viewed as follows:

1. Project Origination

This is the process by which a user and the development organization develop problem statements and decide to proceed to the Project Definition Phase. A project plan is developed to ascertain skills requirements and availability for conduct of remaining phases and a quality review is conducted by development and user management. This is perhaps one of the most important yet neglected phases of the life cycle which will have impact on succeeding phases and influence the degree of success.

Based on a stated need by a user and/or the data processing staff, the systems coordination manager, in conjunction with the appropriate user representative, will develop a general problem statement or objective. The coordination manager identifies user management (project sponsor) who has responsibility in the user organization for development of the project proposal. This objective is submitted to the systems development manager for assignment of a projects manager who in turn identifies appropriate staff levels and skills needed (Project Team) to develop the project proposal.

Allocation of resources and a start date for the project proposal can then be established. This is done by the systems development and systems support managers in consultation with the user functional management, the systems coordination manager, and when appropriate, the Manager of Systems Technology and Consultation. This team (the Systems Management Team) carries authority and accountability within the organization for overall balance between resources and project commitment strategies.

2. Project Definition

At this phase the problem statement is translated to a detailed project plan together with very approximate cost estimates including an upper bound. A decision is made at this time concerning the need to conduct a feasibility or cost/benefit analysis which is normally required.

If a feasibility study is needed, the projects manager will form the team for the feasibility study asking for assignment of other data processing and user personnel as needed. The systems coordination manager, in conjunction with user management (project sponsor), will establish clearly defined focal points for the study within the user community and insure that required user commitment and participation is obtained. However, the final responsibility for the study rests with the systems development organization. At completion of the feasibility study, if continuation is approved by user management, the project enters the systems development phase.

3. Systems Development

During this phase the project proceeds with preliminary design, through detailed design, programming and testing, adhering to accepted standards and guidelines and drawing on necessary skills as required to produce a quality product.

For most large projects, a steering committee is formed. User personnel on the committee should be appointed and charged by user management (project sponsor) in consultation with the appropriate coordination manager(s). The chairman of this committee is from the proper sector of the user community and has the authority to make necessary decisions concerning the project. A projects manager and other members of the management team whose participation would add value are appointed to the steering committee and charged by the Systems Management Team. The steering committee may assign work teams reporting to them during the project life.

The project team is composed of a nucleus of development staff, with other membership, either temporary or permanent, consisting of personnel from within the data processing and/or user community possessing the skills required for the successful execution of the project. This team is named and charged by the development manager together with other data processing and/or user management as required. This charge will clearly delineate the functional responsibility of all team members to the projects manager. The project team will develop and implement the system according to established methodology.

Early in the development cycle the level of operational phase support effort will be agreed upon and the appropriate system development staff will be planned to carry the system into the support mode. Moreover, for most projects an individual(s) functionally reporting to the projects manager and administratively reporting to the support manager will be named and charged to coordinate normal systems support of the project team throughout the remainder of the project.

Attendance of appropriate systems managers is required at periodic project reviews to ensure timely decisions on internal matters and prompt resolution of issues which might otherwise impact project schedules/costs or the plans/operations of other development or user organizations.

4. System Implementation

This phase is the process by which a thoroughly-tested and quality-checked system is placed in a production mode. It includes all user and data processing training and "must do" modifications.

The steering committee will continue to function until implementation is complete. They will approve implementation plans, start and completion dates, provide user liaison and, if needed, provide coordination among user groups and the project team. At the onset of the implementation phase the steering committee and project team will finalize and agree on specific user acceptance criteria.

As implementation proceeds the steering committee will distinguish between "must do" development items and "wish list" support requests. The impact of "must do" items must be clearly understood and accepted by the steering committee. The steering committee will declare the system to be accepted and implemented from the user standpoint. Where "must do" items involve resource commitments significant enough to affect organizational strategies, the Systems Management Team will be consulted.

5. Wrap-Up

This phase consists of the orderly transition of the system to an operational support mode, the reassignment of project team personnel, and the conduct of a project completion analysis and review.

After the system is implemented a determination will be made by the Systems Management Team as to when responsibility for continuing support should move to the support organization. At this time any designated members of the project team will transfer to the support function to handle future modifications. A project completion review is conducted and the projects historical data base is updated to reflect performance of the project by each phase of the life cycle.

E. Estimating Review Board

There are of course many philosophies of estimating project cost ranging from estimating only the cost of the next phase to attempting to furnish a relatively hard estimate for the entire project at project origination time. In most organizations, the user community will press for the best expression of the entire project cost before the project has progressed to any extent. Approaches to preparing a project cost estimate also cover a wide spectrum, but regardless of the approach or philosophy followed, an Estimating Review Board can play an important role in improving the quality of the estimate. It does this by providing a "detached" review of the estimate by experienced people.

The Estimating Review Board should have one or two permanent members plus one to three other members representing the more experienced professional staff and projects managers. The board reviews the estimate for completeness, insuring that there is reasonableness in estimated time and monies to cover all phases and tasks of the project plan. It examines not only the value of the parameters used but possible omissions which may well affect the final project costs. Parameters for which values should be assigned vary from the expression of system size (estimated lines of code, number of programs, number of files, etc.), system complexity, degree of interrelationships, commitment of the user, skills, and experience of the project team and manager.

A very important function of the Estimating Review Board is to "look back and compare". This is accomplished by comparing the present estimate to cost of previously completed systems of similar size, complexity, etc. This type of comparison places considerable importance on the project completion data base concept explained later in this paper.

F. Quality Assurance Function

This function provides development management with an independent assessment of the quality of project plans, adherence to established project management process, and end products or deliverables. Quality Assurance personnel by the very nature of the function itself must walk a very thin line. To be effective, they must remain objective and autonomous, but at the same time function as an advocate, rather than as an adversary, to the projects manager and team. The function must be the responsibility of competent personnel with extensive successful project management experience.

They provide a central project and organizational performance assessment. In addition, they provide input into the development and refinement of standards and guidelines based on data and experience from other projects. Through knowledge gained from past projects, the Quality Assurance function can aid, advise, and provide guidance for projects managers. Assistance can be provided in structuring project plans and determining the degree of management

II. THE MANAGEMENT PROCESS FOR SYSTEMS DEVELOPMENT (Cont'd)

required in accordance with size, complexity and constraints placed upon the project. Additionally, it can serve as an independent sounding board as problems arise, or identify potential problems so that corrective action or alternative strategies can be planned. In general, it can provide counseling based on experience across a large number of projects. It also provides objective evaluation of project status and determines if a project is proceeding according to plan. This is accomplished through impartial reviews and internal walk-throughs.

Included also is a check and balance function within the development process. By independently validating project plans and conducting initial and periodic internal project reviews and quality assurance audits, the Quality Assurance function can identify high-risk areas and suggest possible means to reduce risk, or can recommend contingency plans.

III. MANAGEMENT AIDS AND CONTROLS FOR SYSTEMS DEVELOPMENT

A Project Management Information System is the support mechanism that is designed around the management process itself and brings together all management aids and controls. The individual components, described below, become the framework to manage projects from initiation through completion. They help to manage and control overall projects for successful systems development results as well as to provide information to measure the organizational and individual productivity and performance of the systems analysis and development activities.

A Project Management Information System has the following objectives to support the systems development process:

- Define and communicate objectives
- Plan and estimate resources
- Schedule time and resources
- Select leadership and assign staff
- Track and control progress
- Record, measure and evaluate results
- Report and feedback status
- Control changes
- Provide quality assurance

A schematic of aid and control relationships is shown as an information flow chart on page A-5.

AD-A104 252 INTERNATIONAL BUSINESS SERVICES INC WASHINGTON DC F/G 9/2
US ARMY SOFTWARE SYMPOSIUM (2ND) HELD AT WILLIAMSBURG, VIRGINIA--ETC(U)
1978 S M TAYLOR DAAK70-78-D-0030

UNCLASSIFIED NL

6 OF 1•
AD
A 04252

III. MANAGEMENT AIDS AND CONTROLS FOR SYSTEMS DEVELOPMENT (Cont'd)

A. System Management Methodology

One of the major management tools is a working, viable system management methodology. The methodology can be described as a guide for system development that contains procedures for planning, managing, performing and reviewing the work of systems development projects. It is not a Project Management System in itself, but the cornerstone in a framework for project management - a vehicle, but one that requires adaptation to the organization and type of project.

As systems development efforts become larger and more complex it becomes more important to "steer" a project in the proper direction during the evolution of the system design. This can be done through the standard framework within which all required work can take place in proper sequence. However, within this framework individual work components can vary with end objective.

The methodology describes a "typical" project down to task level. It can guide project leaders toward agreed "best practice" and help avoid overlooking needed work. It serves as a checklist from which tasks can be added or deleted as needed for a specific project. It also provides a common vehicle for communicating about a project: within the project team, within the development organization, and with users. Emphasis is placed on terms of reference, reporting relationships, responsibilities, and commitments at the outset. Standardized documentation of project plans, estimates, and progress as well as end results are provided.

A methodology is a phased approach with positive, discrete events, milestones or checkpoints, and work products. It insures that planning takes place at the beginning of the project itself and each subsequent phase. It also provides for re-estimates at each phase for the next phase and for the remainder of the effort. Timing and manner of project reviews (at checkpoints or milestones) are specified to ensure project work is complete and in compliance with established project management methodology. It makes visible areas of assumption, disagreement, deviation from standards, identifiable uncertainties (risks), and actual or potential slippages/overruns for timely management attention.

It provides for increased user involvement both in the work itself, and in the reviews, to foster awareness and understanding of this critical role in achieving project goals, and to achieve continuity and integrity in translating user objectives into system specifications, and verifying that they are attained. It places emphasis on user and internal communication and commitment, along with control of changes which impact elapsed time and cost. It also emphasizes other internal involvement from operations, auditing, etc.

The methodology can be characterized as a guide (not a straitjacket) that recognizes the legitimacy of planned deviation and provisions for change control. It must recognize that the system development process rarely has clear-cut phase boundaries in the real world and provide the atmosphere for managing variances as a normal course of business. It also must recognize that as a part of a dynamic environment the methodology itself will require constant monitoring and refinement in order to survive.

On page A-6 is a chart depicting typical project development phases and sub-phases of a system management methodology. In Shell, the life cycle is managed with three phases and thirteen sub-phases.

B. Staff Planning System

In the project management process, a key factor to success is the effective management of personnel. To improve project planning and scheduling, relate available skills to workload requirements, avoid underutilization, and provide essential training, a system is required that provides:

- basic skills data
- current and planned assignments
- availability dates of individuals
- training information
- employee personal aspirations

All data provided will be used as input to the overall management and decision process used in departmental, project, and career planning.

Assignments - for each individual, job assignment data will be maintained. This includes past, current and future assignments and will provide type of experience (skill), time, and availability for reassignment information. By including this type of data, forward project requirements and individual aspirations can be considered in the management process.

Training - for each individual, all internal (within the corporation) and external training classes attended are maintained. Also, future training, both normal skill development and individually desired, will be input to the system.

The results of these efforts in the area of staff planning should help in the overall objective of improving the project management process and providing meaningful appropriate career path assignments based upon business needs as well as personal aspirations when possible.

C. Project Control and Tracking System

Another key management aid is a simple, straightforward, friendly system for project planning, control and tracking. Such a system for project reporting, estimation, and completion history allows for entry and tracking progress of work plans and allows updating of work plans through approved change control procedures.

Project work-plan tasks and estimates are compared to actual expenditures and completions by project personnel. The level of comparison is optional but can include sub-project, program, and task. A provision to identify phases of the development life cycle is included. Control reports track plan vs. actual for starting and completion dates and manpower expended. Exception reports isolate late completion dates and manpower utilized in excess of approved projections. Reporting periods are approximately every two weeks and the accumulation of historical performance aids management in more accurate estimating of future projects while serving as a measure of performance to plan on a current basis.

D. Project Costing System

This system accumulates all project costs by phase, including manpower, direct expenses, resources and overhead. The system produces cost and control reports for management and provides current and cumulative data useful for tracking manpower and data processing resources expended compared to approved plans.

Project computer usage is captured automatically at execution time. Manpower costs are input automatically from the Projct Control and Tracking System mentioned above. Other expenses are automatically captured from appropriate source documents.

E. Change Control System

Ongoing change is inevitable during any project development cycle, and the type, amount, and time of a change can have a significant effect on the successful completion of a project. Furthermore, whether the changes are a result of better understanding of the user's business requirements, new laws or regulations, new technology, changes in procedures, etc., failure to manage change as an ongoing process will guarantee overexpenditures, missed target dates, or a product which does not meet user expectations.

III. MANAGEMENT AIDS AND CONTROLS FOR SYSTEMS DEVELOPMENT (Cont'd)

The effective control of changes, particularly for large projects, can only be accomplished through consistent application of a well-understood and managed control procedure. Although change control should be tailored to best meet the needs of the organization it serves, the following characteristics should be considered as minimum requirements:

1. Documentation

   Change control procedures should be formalized in a written document to provide common understanding to all affected parties.

2. Controllable Change

   Agreement should be reached as to what constitutes a change that will be controlled using the procedure.

3. Identification

   Recognition of the fact that a change has occurred requires complete documentation of all agreements on requirements between developers and their users. The procedures should emphasize that

   these documented agreements will serve as the sole basis for identifying and evaluating change.

4. Evaluation

   Each change must be evaluated in objective terms with respect to its need, criticality, timing, cost/benefit and scheduling effect on the project.

5. Approval

   The project steering committee, or similar body, will be responsible for approval/disapproval of all requested changes.

   The committee will normally be comprised of both development and user representatives and will base its decision on the completed evaluation of the change. The final decision is solely vested with the user.

6. Notification

   A formal communication process must be available for requesting changes, and for notification to affected parties of the results and impact of the change request.

III. MANAGEMENT AIDS AND CONTROLS FOR SYSTEMS DEVELOPMENT (Cont'd)

7. Records

Formal and complete records must be kept, by project, of all changes; this includes a summary of cumulative impact over time of changes on each project.

F. The Quality Review Process

Personnel are appointed to a project Quality Review Board to represent all interested/affected systems groups and users for the purposes of concurring with project work completed at each of the agreed milestones during the development cycle. The Board's effectiveness is enhanced by appointing one or two members as "evaluators" who review the deliverables in detail and present their findings to the Board. The Board's responsibility is not to "approve" or "disapprove" per se, but to highlight any areas of disagreement or risk for appropriate management attention and action. The number of reviews is flexible, ranging from 4 to 10 depending on project size and complexity.

G. Project Completion Data Base

A necesary support element in a Project Management Information System for the development organization is an historical record of data from completed projects to assist in planning, managing, and estimating future development activities. Typical data elements include size, duration, complexity, cost, performance information by life cycle phase, assessment of degree of success and user satisfaction level, tangible economic benefits, degree of compliance with standards and guidelines, productivity tools utilized, etc.

IV. MANAGEMENT SKILLS FOR SYSTEMS DEVELOPMENT

Successful systems development requires a high degree of business, technical, and general management skills and experience which must be developed and maintained. The following identifies those considered to be most significant.

A. Project Management

Required to manage the end product in the major areas of quality control, tracking, and people to reach the objectives of on-time, within budget and to user satisfaction.

IV. MANAGEMENT SKILLS FOR SYSTEMS DEVELOPMENT (Cont'd)

B. Communications

Allows the manager to be effective in giving and receiving essential information when interfacing with users, subordinates, upper management, and other development or support groups.

C. Team Building

Focuses on strengths of individuals, group dynamics, project team objectives, sharing, job enrichment, etc.

D. Business Management

This category is used to identify technical and professional tools or techniques that support the managerial process. Examples include problem solving and decision-making techniques, simulation techniques, planning and forecasting tools, financial analysis.

1. Financial Analysis

Develops skills in cost/benefit analysis to identify real costs, real benefits, tangible/intangible value, etc.

2. Problem Solving and Decision-Making

Presents the manager with a systematic, quantitative approach to the decision-making process in selection of imperatives, desires, risk analysis and regret analysis.

3. Planning Methodology

Provides the manager with methods and tools for short- and long-range planning of information systems and includes

- understanding the business and processes
- identifying business and information requirements
- defining information systems network.

E. Leadership

Leadership skills are those skills that are concerned with organizing and directing the work of others. Leadership is oriented toward the achievement of specified task objectives and does not, as defined here, include the broader administrative functions of business and personnel management.

This provides an understanding of leadership styles to assist managers in choosing a leadership pattern that will be the most effective for their organization.

F. Technology

Technical skills can be broadly defined as data processing skills to include the technical specification or usage of programming languages, data base management systems, file management sytems, general utility software, micrographics, telecommunications, input/output media, systems software, and systems analysis and design techniques. Following are major areas of focus for management skills development.

1. Application Software Development

Managers must be kept current on software packages available that will improve productivity and have an understanding of how software is actually developed.

2. Data Analysis and Data Base Design

Techniques of data analysis and data base design and selection criteria for appropriate Data Base Management Systems is required to produce the most cost effective systems that optimally meet data storage and retrieval/access requirements and provide flexibility for future requirements.

3. Telecommunications

Managers must be aware of current and future telecommunications software and hardware technology to properly direct the systems design effort to take advantage of improved methods and avoid near-term obsolesence.

4. Other

Managers must be familiar with information systems philosophies and technologies such as distributed data processing, electronic funds transfers, electronic mail, word processing, operating systems, micrographics, optical character recognition, security and encryption, backup recovery and contingency, etc.

G. Staff and Organizational Performance and Productivity Measurement

Managers must measure and evaluate individual and organizational performance, proposals, tasks, tools, etc., to improve the overall performance and productivity of their project teams.

IV. MANAGEMENT SKILLS FOR SYSTEMS DEVELOPMENT (Cont'd)

H. Business Function Knowledge

Functional (user) knowledge is a term used to identify the primary operating and service departments of a corporation (e.g., Financial, Marketing, Manufacturing, Purchasing, etc.). The reference to functional knowledge implies an understanding of a function's activities, the relationships between the activities and a firm grasp of systems and procedures that exist or are required to support them.

This is needed by managers to understand how and where their segment of the business fits into the overall business function's objectives and goals.

I. Estimating and Risk Assessment

These skills are essential in the planning and controlling aspects of project management. The risk assessment is an important ingredient in a "go/no go" implementation decision.

J. Staff Development and Administration

Personnel development and administration embraces salary administration, employee evaluation, career planning, and employee communication and counseling.

Where practical, managers must provide work assignments that reflect and magnify the employee's strengths and aspirations, provide training opportunities for technical and professional development, and afford their staff the appropriate growth opportunities and career plans.

K. General Business Skills

General business is used to describe those skills that are derived from a knowledge of broad-based subjects that can have a significant impact on business activities. Typical subjects are governmental policies or controls, company social awareness programs, principles of business and finance, etc.

V. STANDARDS AND TRAINING NEEDED FOR SYSTEMS DEVELOPMENT

A. Areas Requiring Standards, Guidelines and Advisories

The term "standards" has many meanings and evokes highly subjective (if not emotional) reactions. It is useful to define several categories and to note that it is possible to allow

V. STANDARDS AND TRAINING NEEDED FOR SYSTEMS DEVELOPMENT (Cont'd)

flexibility of application within even the most restrictive category. For example: It is a standard that all projects will follow the prescribed System Management Methodology. That means that no project may arbitrarily pursue a totally different course; however, within the methodology there are many options available to match the tasks, review points, and work products to the requirements of individual projects. Suggested definitions are:

Standard - a conceptual, procedural or technical discipline which is mandatory and must be followed unless exception is specifically authorized by the appropriate (relatively high) level of management.

Guideline - a conceptual, procedural or technical practice that is intended to assist in accomplishing work. They are to be followed in the absence of specific reasons for deviation (which must be documented and justified at review points).

Advisory - a conceptual, procedural or technical advice of agreed merit promulgated for guidance and assistance. Compliance is voluntary.

For standards to be meaningful, specific enforcement facilities must exist. Preferred methods are computer assisted and built into the application development/maintenance process. When this is not feasible, then manual procedures must be used. The key is establishing clear responsibility for standards enforcement and a set of structured procedures for monitoring conformance to standards.

The cornerstone of an effective standards program is a visible, workable mechanism for identifying needs, locating expertise and responsibility, obtaining reviews and approvals, publishing and cataloging. Euthanasia (mercifully killing obsolete or moribund standards) must also be provided. (See chart on page A-7 for application of standards in life cycle.)

1. Vendor Software

This is an important area because of the currently growing availability of purchased software as an alternative to in-house development, which is becoming more costly over the entire system life. Evaluation criteria for selection among competitive products should be established. The specific areas of use for new software should be defined.

2. Programming Languages

The standard set of supported languages must be defined. Criteria should be established for selection of the appropriate language for a given application. Standards are necessary for structure and coding within a specific language.

3. File Management Systems

Similar to programming languages, criteria for selection among available alternatives and for structure and coding are required. Additional standards should be established for use of various data structures supported by the file management system.

4. Data Base Management Systems

Standards required are similar to those for file management systems. However, because of increased complexity, integrated (shared) data bases, and usual teleprocessing access, more standards are required and must be more rigidly enforced. Some degree of centralization of the data administration function is necessary for control and enforcement of standards.

5. Interactive Programming Facilities

Standards should be established for selection of interactive versus batch facilities. Also for selection of a specific interactive facility for systems development. Acceptable performance levels should be defined. Standards are required regarding when and how to use specific interactive commands or procedures. Security standards must be developed and enforced.

6. Data Dictionary/Directory

Standards are required for which data and applications must use the dictionary. Minimal information required regarding data or program elements and relationships must be defined. Responsibility for updating the dictionary must be assigned and controls and security established. Standard fixed and optional reports must be developed. Standards must be developed and enforced for using the dictionary as the only source for data definitions with program data areas generated only from the dictionary.

V. STANDARDS AND TRAINING NEEDED FOR SYSTEMS DEVELOPMENT (Cont'd)

7. Naming Conventions

Naming conventions are particularly important if a data dictionary is to be an integral part of application development and maintenance. Naming conventions are required for:

- Jobs (including JCL elements)
- Programs and subroutines
- Program elements (procedures, data areas)
- Data sets
- Data elements and groupings.
- Projects and Phases

8. Design of Data Structures

Standards are required for both logical and physical data structures. The objectives are to develop efficient structures while maintaining flexibility for future uses. These standards are especially important for the complex structures supported by the data base management systems.

9. Code Generation and Generalized Programming Facilities

Data dictionary and in-house software are used to reduce programming time and enforce standards through generation of code. Partial or complete programs can be generated from specification forms. Data descriptions can be generated from a data dictionary. Video display terminal screen format definitions are generated from specification forms.

Use of table and macro-driven code generation software can reduce programming time and enforce standard techniques and formats. General trade-offs are reduced personnel costs versus increased computer resource requirements and limitations of functions performed. The trend of rising personnel costs should lend to increased use of generalized code, generated code, generalized programming facilities, and problem statement languages.

10. System Acceptance

Specific criteria should be defined for determining if a new or revised application is ready for production status. Areas to be considered are:

- documentation
- integrity and performance test results
- conformance to standards
- conformance to user's specifications
- provision for back-up, recovery, and contingency

11. Security

Levels of exposure should be defined for both unauthorized access and update of each data set. Standard security measures should be provided dependent upon the level of exposure. Standard security monitoring procedures are also essential.

B. Training of Professional Staff

To maximize the use of new technology, related training is needed at early stages in advance of initial installation. The availability of training as project needs arise helps realize the potential of dynamic computer technology. Training coverage must adapt to complement new development support tools (i.e., preprocessors, new compilers, new macros that isolate the programmer from the intricacies of DBMS and TP) and new development and productivity methods. For most of the staff new training and retraining are subsets of complete training that is required only by a small group of specialists. Many of the development staff work in high level languages that support structured programming, consult with skilled specialists, and are trained in basic/intermediate concepts of the languages that comprise the family of codes. Classroom materials require effort to be up-to-date and manuals must be oriented to current standards and procedures.

The training staff should develop and conduct on site classroom courses to meet the technical requirements of:

- All systems professionals in the uses of new software and hardware technology and techniques.

- College trained employees, other new hires, and transfers in basic programming skills and systems design fundamentals.

- Projects managers, leaders, and team members in Project Management and Systems Management Methodology.

- User groups who communicate with the computer (such as engineering, financial, marketing, service groups, etc.).

- Other computer-related functions which interface with systems groups (such as operations, data control, scheduling, user services, etc.).

V. STANDARDS AND TRAINING NEEDED FOR SYSTEMS DEVELOPMENT (Cont'd)

- Non-data processing staff who need familiarization with computer concepts and terminology.

Technology to be applied in project development to satisfy user objectives creates an intensive challenge for staff training. To meet this challenge we must attain an equitable balance between training cost, training needs and training availability. We have found on-site classroom training desirable for basic entry level courses and courses with large attendance. Media-based instruction provides training for refresher courses, quick orientation, expanded offerings, and individual flexibility.

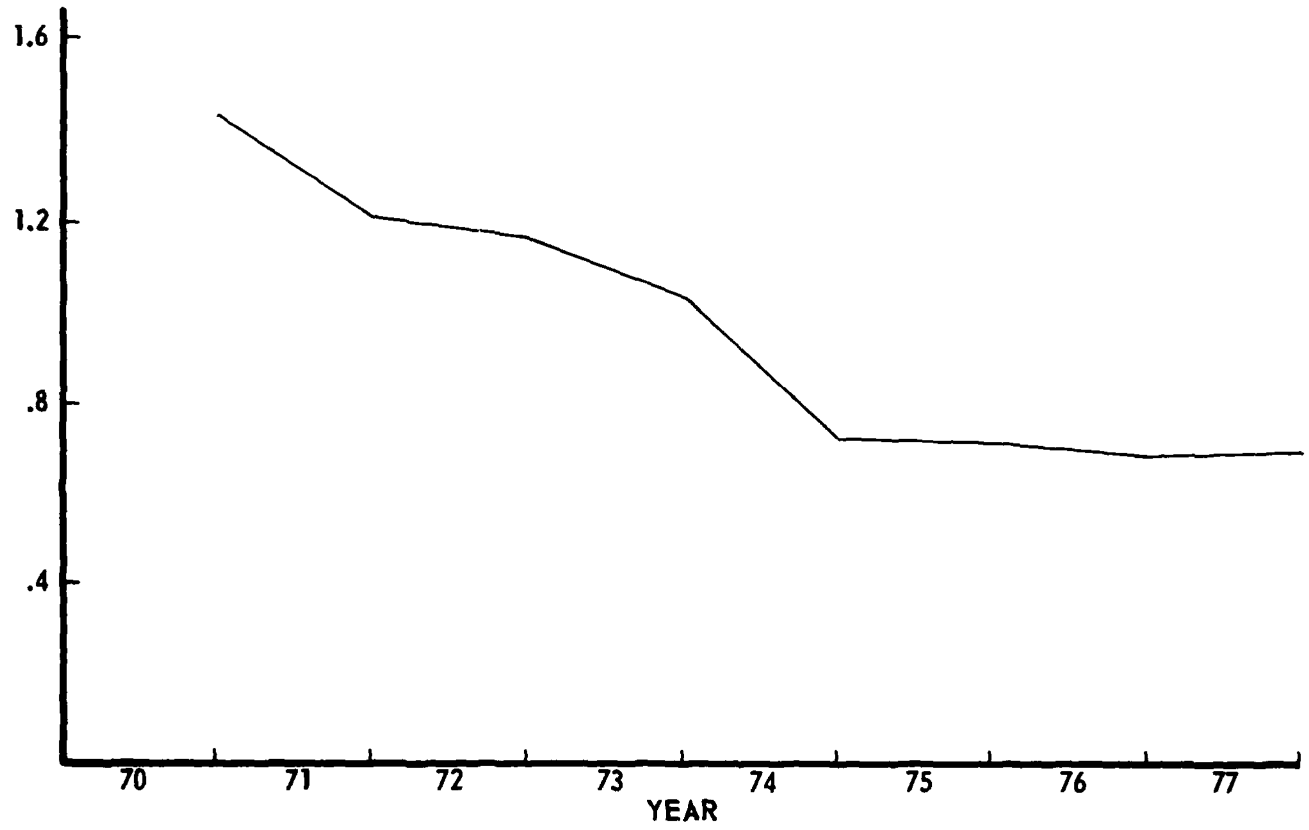
VI. PROJECT COMPLETION ACTIVITIES

Project completion should be a joint activity between user, support and development management. It is a planned and formal event of the management process. It should be a discrete event which marks the termination of development activities. Major activities at this post completion review and audit include:

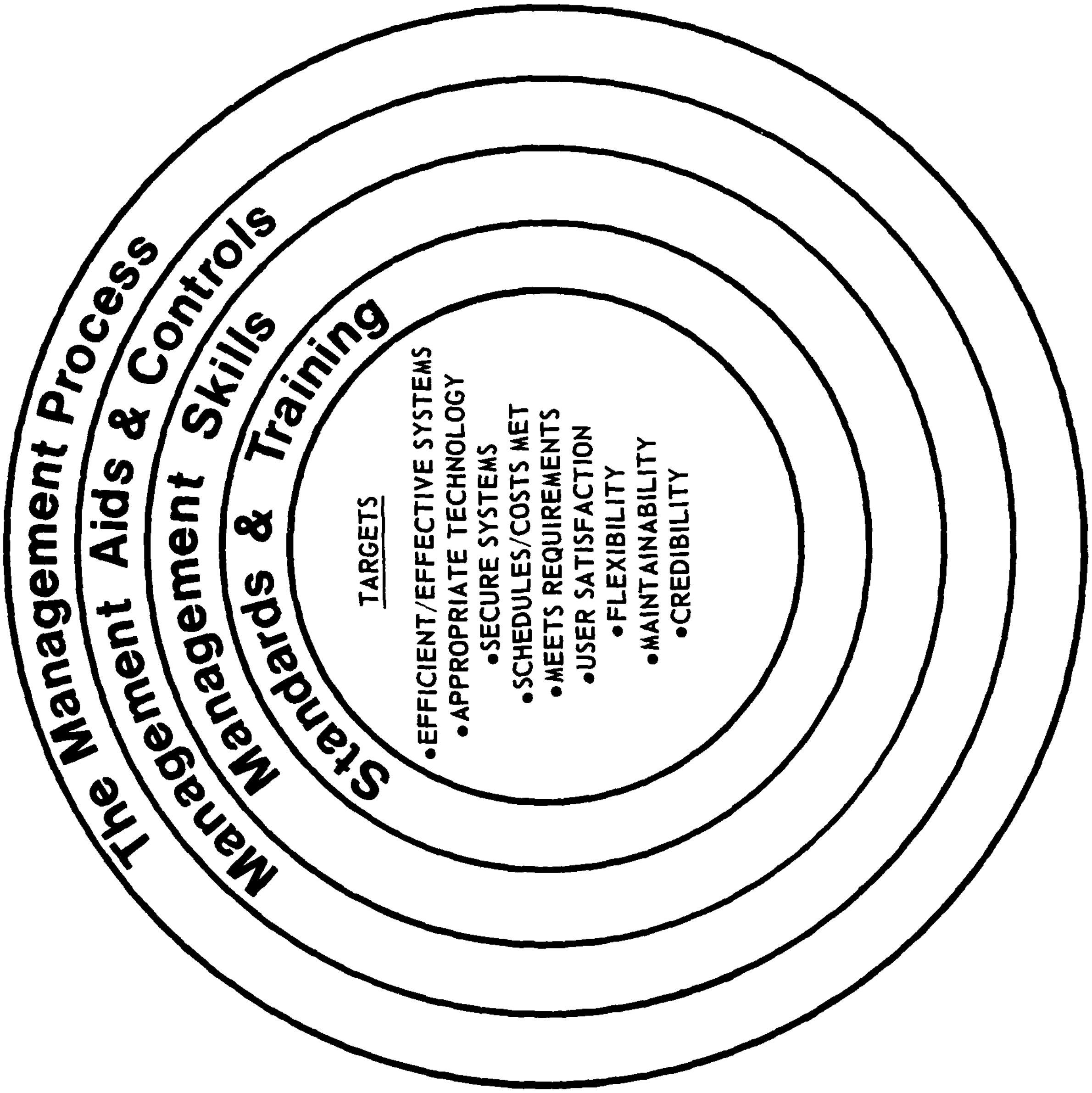
- Evaluation, grading and documentation of project team performance, end product quality, and level of user satisfaction.
- Feedback to project management and staff.
- Review of problems encountered during conduct of the development process in order to minimize future occurences.
- Review of usage and degree of success with productivity techniques, development methodology, and management processes.
- Formulation of appropriate recommendations for enrichment of standards, guidelines, and advisories.
- Update of factors, parameters, and other information to enhance estimating activity.
- Update historical project data base by recording size, duration complexity, cost, performance information (by life cycle phase), assessment of degree of project success, tangible economic benefits, degree of compliance with standards and guidelines, productivity tools utilized, etc.

APPENDIX A

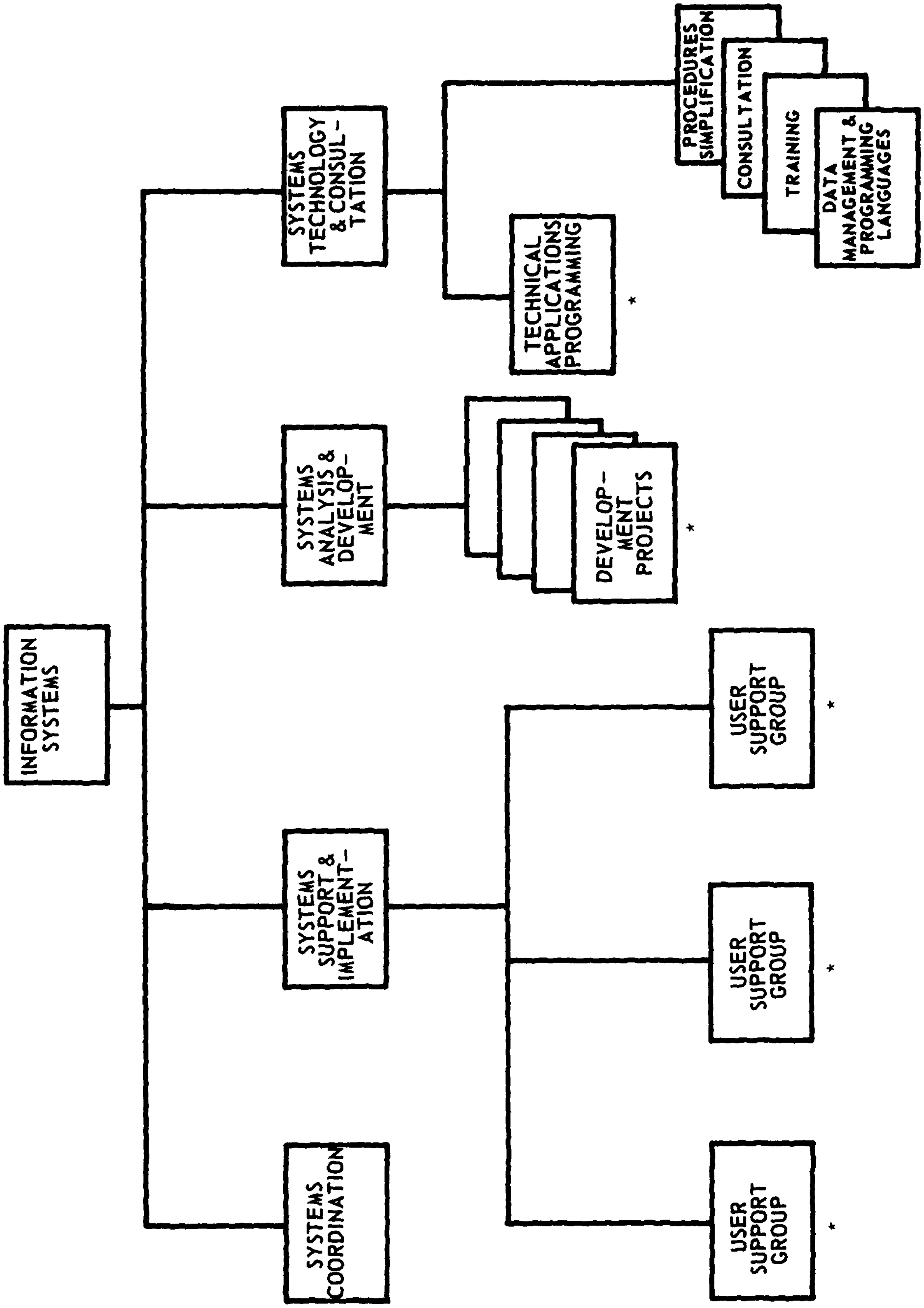
DP EXPENSE AS A % OF REVENUE



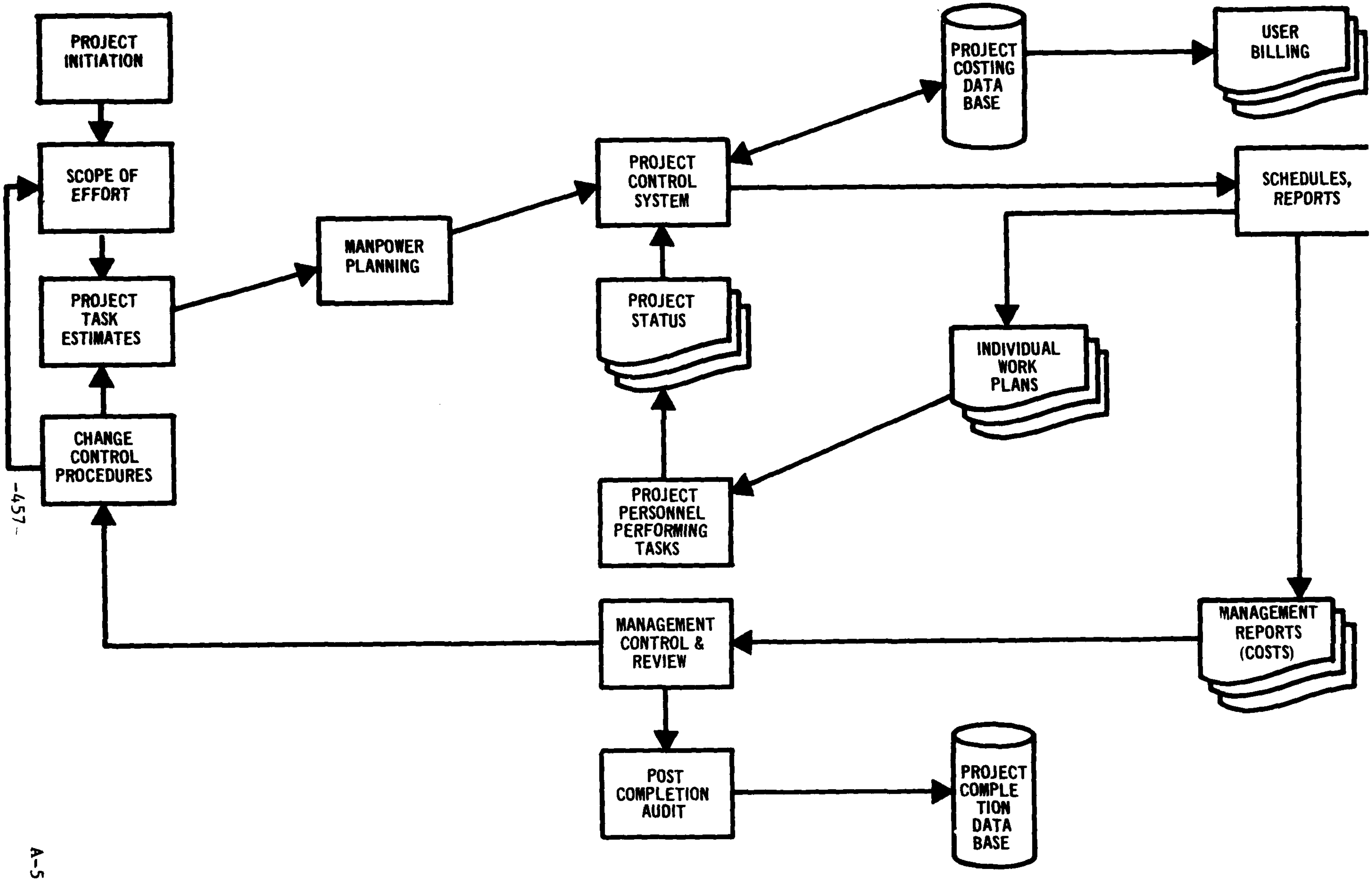
A-1


The Management Process
Management Aids & Controls
Management Skills
Standards & Training
TARGETS
•EFFICIENT/EFFECTIVE SYSTEMS
•APPROPRIATE TECHNOLOGY
•SECURE SYSTEMS
•SCHEDULES/COSTS MET
•MEETS REQUIREMENTS
•USER SATISFACTION
•FLEXIBILITY
•MAINTAINABILITY
•CREDIBILITY

THE INFORMATION SYSTEMS DEPARTMENT

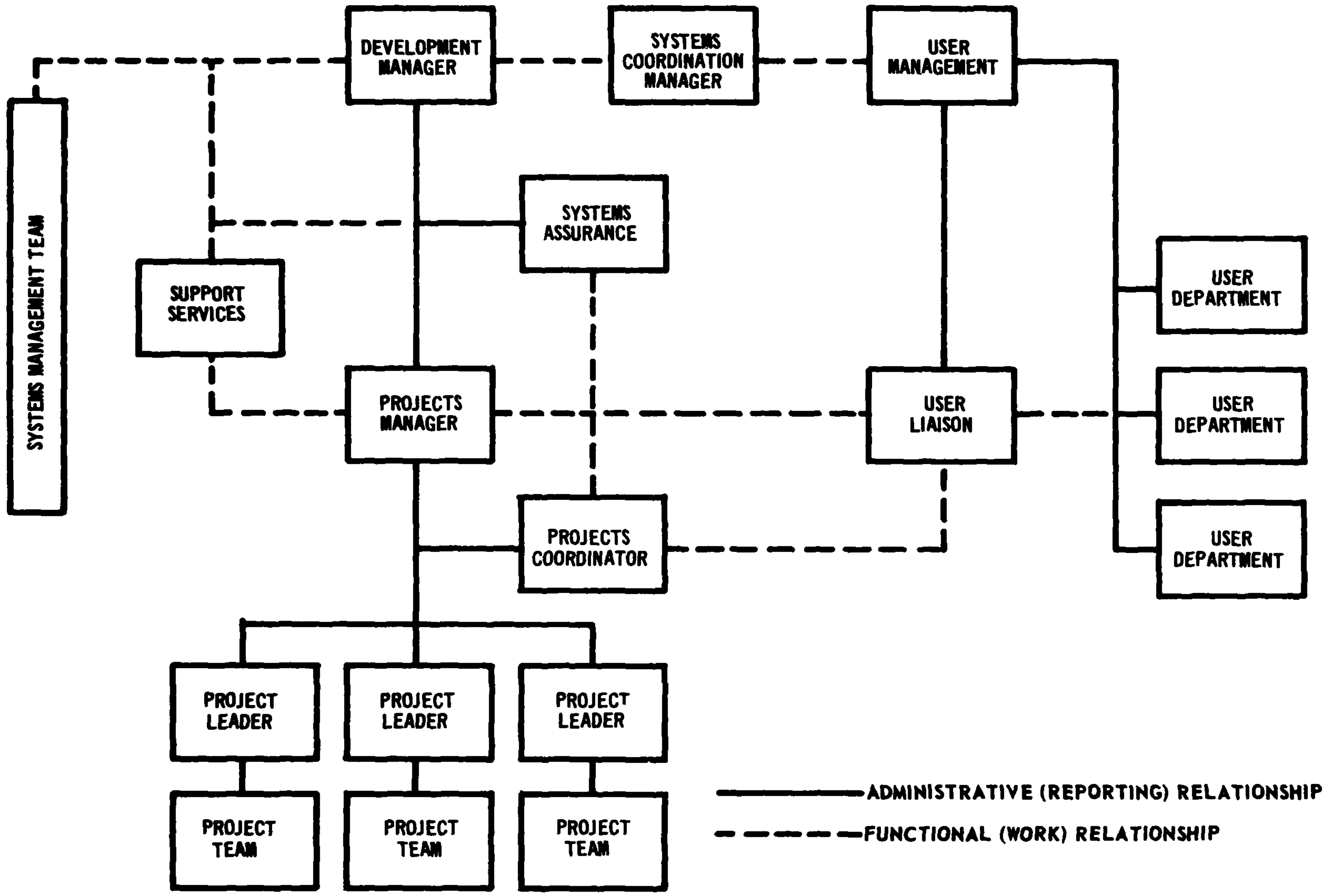

INFORMATION SYSTEMS
SYSTEMS COORDINATION
SYSTEMS SUPPORT & IMPLEMENT-ATION
SYSTEMS ANALYSIS & DEVELOP-MENT
SYSTEMS TECHNOLOGY & CONSUL-TATION
USER SUPPORT GROUP *
USER SUPPORT GROUP *
USER SUPPORT GROUP *
DEVELOP-MENT PROJECTS *
TECHNICAL APPLICATIONS PROGRAMMING *
PROCEDURES SIMPLIFICATION
CONSULTATION
TRAINING
DATA MANAGEMENT & PROGRAMMING LANGUAGES

SYSTEM MANAGEMENT AIDS AND CONTROLS



- DEFINE AND COMMUNICATE OBJECTIVES•PLAN•ESTIMATE
- SCHEDULE (TIME & RESOURCES)•SELECT LEADERSHIP & ASSIGN STAFF
- TRACK AND CONTROL•MEASURE & EVALUATE RESULTS
- REPORT & FEEDBACK•CONTROL CHANGES•QUALITY ASSURANCE

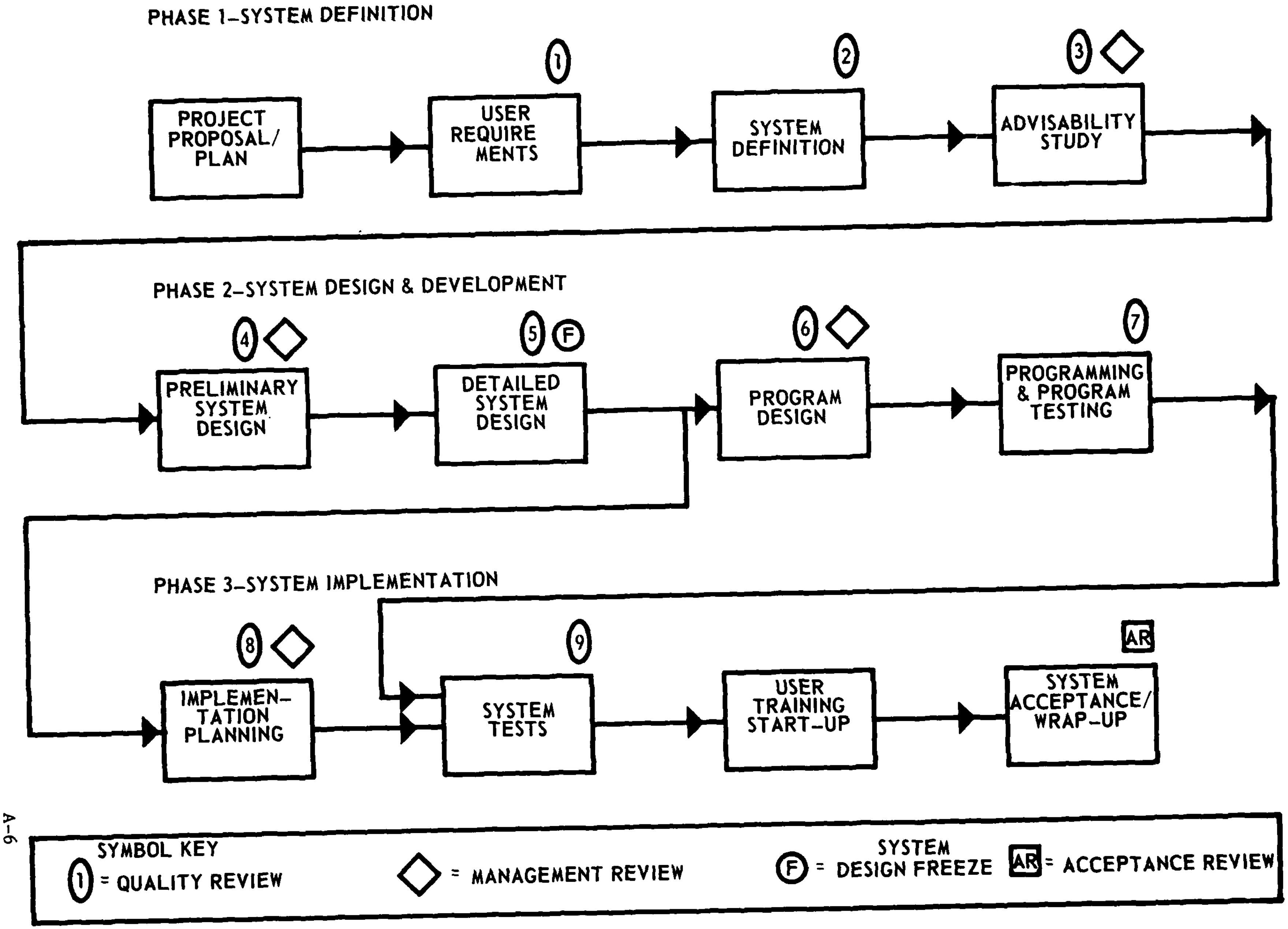
PROJECT DEVELOPMENT ORGANIZATIONAL RELATIONSHIPS



APPLICATION OF STANDARDS TO SYSTEMS LIFE CYCLE

	SYSTEM DEFINITION				SYSTEM DESIGN AND DEVELOPMENT				SYSTEM IMPLEMENTATION				
	PROJECT PROPOSAL/ PLAN	USER REQUIRE-MENTS	SYSTEM DEFINI-TION	ADVISA-BILITY STUDY	PRELIMINARY SYSTEM DESIGN	DETAILED SYSTEM DESIGN	PROGRAM DESIGN	PROGRAMMING & PROGRAM TESTING	IMPLEMEN-TATION PLANNING	SYSTEM TESTS	USER TRAINING START-UP	SYSTEM ACCEPTANCE WRAP-UP	SYSTEM OPERATIONS/ SUPPORT
System Management Methodology	X	X	X	X	X	X	X	X	X	X	X	X	X
Programming Languages							X	X					X
File Management Systems					X	X	X	X					X
Data Base Management Systems					X	X	X	X	X				X
Interactive Programming Facilities								X					X
Documentation	X	X	X	X	X	X	X	X	X	X	X	X	X
Data Dictionary/ Directory					X	X	X	X	X	X	X	X	X
Design of Data Structures					X	X							
Naming Conventions					X	X	X	X					X
Execution Procedures							X	X	X	X			
Generalized Code						X	X	X					
Testing/Debugging					X	X	X	X	X	X			
Code Generation							X	X					
Generalized Programming Facilities (ADF)							X	X					
Enforcement Facilities								X		X		X	X
System Acceptance										X	X	X	
Security		X			X	X	X	X	X	X	X	X	X

SHELL MANAGEMENT METHODOLOGY
PROJECT DEVELOPMENT LIFE CYCLE



HARDWARE/FIRMWARE/SOFTWARE TRADEOFFS

Dr. Serafino Amoroso
CENTACS

HARDWARE/FIRMWARE/SOFTWARE TRADEOFFS

SESSION CHAIRPERSON: Dr. Serafino Amoroso

Software Engineering Division
CENTACS

SESSION SUMMARY

In the context of real-time software support, one of the main advantages of microprogramming is in extending the capabilities of an existing mainframe CPU either for the benefit of the operating system or for individual applications. It is a cheaper or more efficient method of implementing certain functions; traditionally implemented in hardware or software, as well as aiding in system maintenance by gathering statistics on performance. This session has been concerned with the tradeoffs that can be made between hardware, software and firmware. The first (1) paper described an emulator which was developed as a form, fit and function replacement for the AN/GYK-12 computer. The second (2) paper described an effort in which a smart peripheral driven by dedicated firmware was used to provide technology insertion upgrade without disrupting the operational software. The third (3) paper described the software/firmware development approach taken on a small experimental missile computer with no available support software.

TACTICAL AN/GYK-12 EMULATOR, Edward J. Beach, U.S. Army CORADCOM, CENTACS (System Validation Division) Fort Monmount, N.J.

TECHNOLOGY UPGRADE OF EXISTING SYSTEMS PERIPHERALS, Jeffrey S. Yohay and Martin I. Wolfe, U.S. Army CORADCOM, CENTACS (Software Engineering Division), Fort Monmouth, N.J.

A CASE STUDY OF THE SOFTWARE/FIRMWARE DEVELOPMENT FOR A MICROPROCESSOR-BASED COMPUTER, James E. Scott, U.S. Army MIRADCOM, Missile System Software Center, Redstone Arsenal, Alabama

A Tactical AN/GYK-12 Emulator

Edward J. Beach

CENTACS

A tactical emulator has been developed which is a form, fit, and function replacement for the AN/GYK-12 computer as used in TACFIRE and TOS[2]. The emulator combines the CPU-IOU into one "A" size case and provides 131K of core memory in each Emulator Mass Core Memory Unit (EMCMU). The emulator has been demonstrated in a TACFIRE shelter and runs all TACFIRE software unaltered. A compute and execute TACFIRE fire plan was run on the emulator one-third faster than on the existing AN/GYK-12. Since the emulator is microprogrammable, instruction alteration and addition are easily accomplished. Built-in firmware fault isolation diagnostics allow faults to be isolated to one of the twelve replaceable modules in each EMCMU.

A TACTICAL AN/GYK-12 EMULATOR

Edward J. Beach

US Army Communications Research
and Development Command
Center for Tactical Computer Systems
Systems Validation Division
Fort Monmouth, New Jersey

INTRODUCTION

In response to direction by the TACFIRE Army Systems Acquisition Review Council (ASARC) to provide business competition for the AN/GYK-12 computer used in TACFIRE, a tactical emulator has been developed which is a form, fit, and function replacement for the AN/GYK-12 computer. The emulator has been demonstrated in a TACFIRE shelter and runs all TACFIRE software unaltered. Programs are executed approximately one-third faster than on the existing AN/GYK-12. Built-in firmware fault isolation diagnostics allow faults to be isolated to one of twelve replaceable modules in the emulator CPU/IOU or to one of twelve replaceable modules in each emulator Mass Core Memory Unit.

The advantage of utilizing an emulator to replace an existing computer includes the capability to use new technology to replace components that are out-dated and thereby increase reliability, availability, and maintainability. Decreased cost, size, weight, and power and increased speed may also result. No software expense is incurred since all the software is retained. A significant advantage provided by an emulator is the capability to modify the instruction set of the emulated machine easily and inexpensively.

CONTRACTOR REQUIREMENTS

Control Data Corporation of Minneapolis, Minnesota, was awarded a contract by the Communications Research and Development Command (CORADCOM), formerly the US Army Electronics Command (ECOM) to design and build two prototype AN/GYK-12 emulators for use in a TACFIRE battalion configuration.

Each emulator consists of one emulator CPU/IOU (ECPU/EIOU) and three emulator Mass Core Memory Units (EMCMU's). The emulator's CPU/IOU and memories are required to be a form, fit, and function replacement for the AN/GYK-12 CPU/IOU and MCMU's as used in TACFIRE. They are further required to provide full transportability of software, meet or exceed current AN/GYK-12 performance, obtain identical logic results, drive existing TACFIRE peripherals, and provide, as a minimum, the same expansion capabilities as the AN/GYK-12.

A demonstration that these requirements were met is being given by installing the emulator in a TACFIRE system in the field and performing the TACFIRE function through test scenarios and artillery training fire missions.

The TACFIRE system is used by the Field Artillery to increase artillery efficiency, expand its effectiveness, and expedite command decisions through automation. TACFIRE consists of tactical equipment used by operators in the field to aid the commander in delivery of fire. The TACFIRE system consists of priority driven multi-task operations which perform the following functions: fire missions, fire planning, fire control, target intelligence, target analysis, ammunition and fire unit status, and meteorological data processing.

EMULATOR HARDWARE

The AN/GYK-12 emulator CPU/IOU and emulator Mass Core Memory Units replace the CPU, IOU, and MCMU's in TACFIRE. The characteristics of the emulator and of the AN/GYK-12 include:

a. thirty-two bit instruction word;

b. one, eight, sixteen, thirty-two, or sixty-four bit data word;

c. memory expandable in 131K MCMU's to a maximum of 2M words;

d. memory access control and protection for program and I/O separately and internal parity checking;

e. one-hundred basic instructions plus 50 extended mnemonic instructions;

f. nine addressing mode combinations;

g. sixty-four program levels;

h. sixteen 32-bit general purpose registers and 16 page registers per program level and special purpose registers;

i. program initiated, but independently operating, I/O data transfers at rates up to 400,000 words per second. A queue word for each program level provides stacking of interrupts in automatic priority and high speed multi-program switching in single or multiple processor configurations.

The EMCMU has a cycle time of 1.2 microseconds, which is slightly over twice as fast as the cycle time of the AN/GYK-12 MCMU. The emulator memory interface is based on Control Data's 480 computer memory interface. Therefore, one cannot replace the AN/GYK-12 MCMU with the EMCMU without also replacing the CFU and IOU with the ECPU/EIOU. Similarly, if one replaces the CPU and IOU with the ECPU/EIOU, he must also replace the MCMU's.

The ECPU/EIOU is enclosed in one TACFIRE "A" size case, taking half the space of the AN/GYK-12 CPU and IOU which require one "A" case each. The emulator enclosure and C-frame together serve as a carrying case and allow the ECPU/EIOU to be rack mounted in the shelter. Equipment slides on the top and bottom of the C-frame allow the enclosure to be extended out of the C-frame for access to the two Power Supply Modules (PSM's) mounted in the rear half of the enclosure. The front panel is hinged and may be opened to gain access to the logic modules which are plugged into receptacles on the wire-wrap plate. The power and signal connectors are located on either side of the front panel.

The front panel of the emulator CPU/IOU is shown in Figure 1. The eighteen Diagnose Status Register lights which are used to display error conditions on the AN/GYK-12 CPU's front panel are condensed into a six digit octal display on the front panel of the emulator. The AN/GYK-12 IOU's nine indicator lights for Data Exchange, Device Channel, and Memory errors are condensed into a three digit octal display on the emulator. This space saving allows room for the remaining indicators and switches of the CPU and IOU, and for additional controls to run and monitor firmware diagnostics.

The enclosure of the EMCMU is similar to that of the ECPU/EIOU. The EMCMU has only one PSM in the rear of the enclosure. Beneath the PSM are two doors for access to four Core Memory Modules (CMM's). The remaining four CMM's and the logic modules are behind the front panel.

Each of the logic modules in the ECPU/EIOU and EMCMU is either a single or double card assembly consisting of one or two ten layer printed wiring boards bonded to a heat sink with either a 152 or 228 pin connector mounted on the bottom edge. Up to 56 integrated circuits are mounted on the surface of each board. Test points at the top edge of the printed wiring boards are used for interconnections between the two boards. Ramp-clamp retainers at the outside edges of the heat sink secure the module in the enclosure and conduct heat to the cooling fins. The single circuit card assemblies have a connector on the top of the module used to bring the signals to the front panel and its connectors.

EMULATOR ARCHITECTURE

Functionally the ECPU/EIOU, as depicted in Figure 2, is comprised of the emulator CPU processor (ECPU), the emulator IOU processor (EIOU), the external buses, the operator/status panel, and the power supplies.

The ECPU, whose design is based on Control Data's Tactical Microprogrammable Processor (TMPP), consists of four modules: the Micromemory Module (MMM), the Processor Control Module (PCM), the Arithmetic Logic Module (ALM), and the I/O Module (IOM). The MMM contains the firmware which emulates the instructions

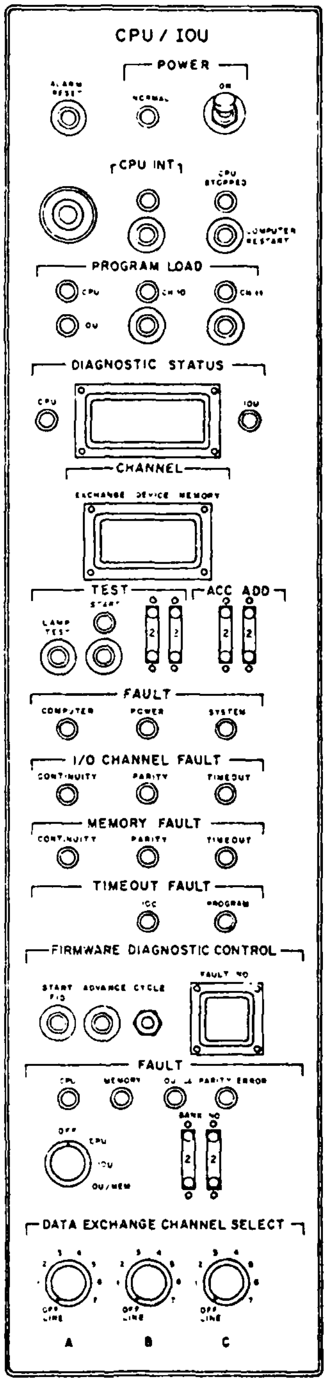


FIGURE 1. EMULATOR CPU/IOU FRONT PANEL

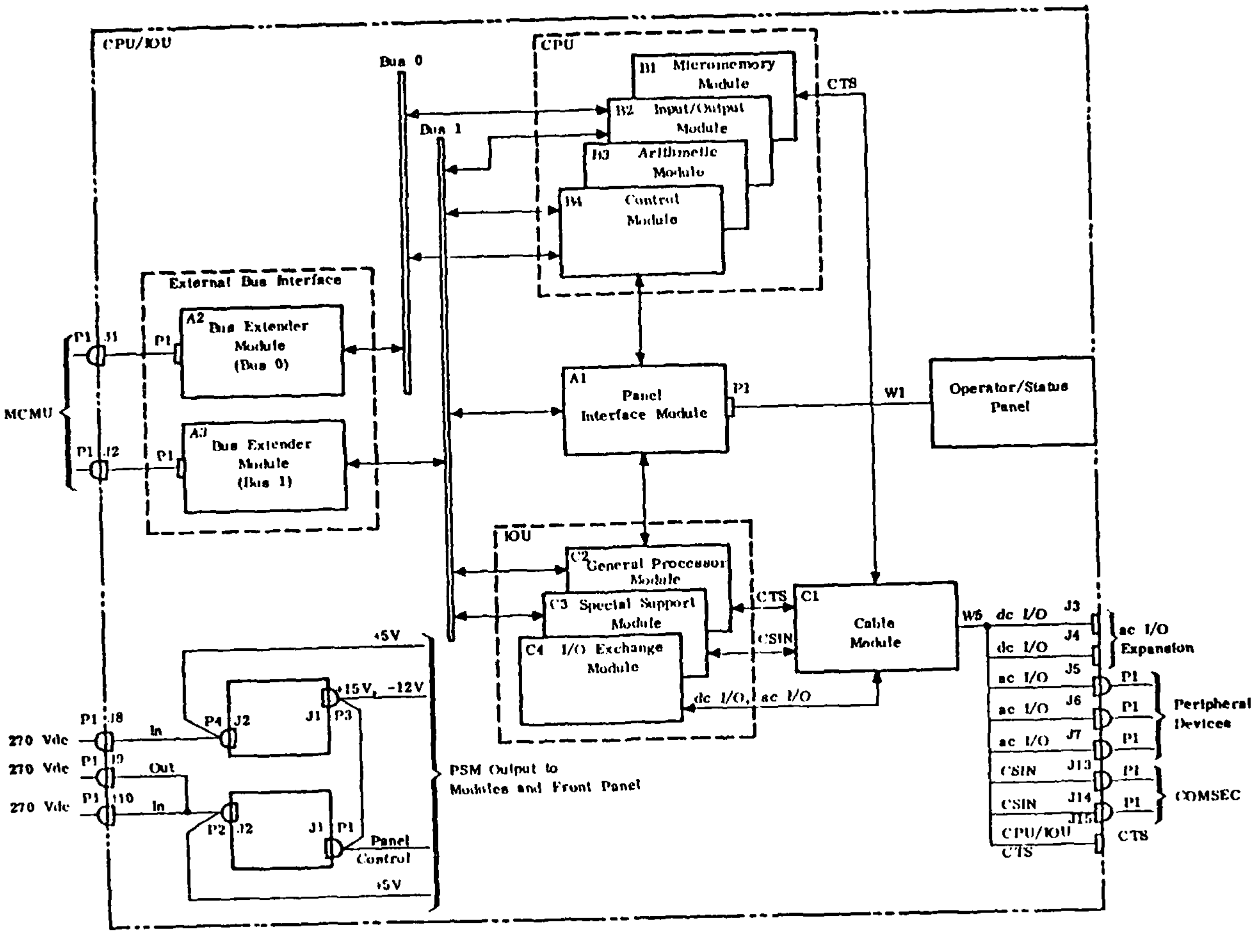


FIGURE 2. EMULATOR CPU/IOU BLOCK DIAGRAM

of the AN/GYK-12. The PCM decodes the AN/GYK-12 instruction for operand and operand address data and generates microtransform data to the ALM to select firmware to be executed. In the ALM, various signals, including the contents of file registers and microtransforms, can be selected for routing to other parts of the ECPU. The ALM also contains an adder which allows boolean functions and shifts to be performed on data before it is routed through the ECPU. The IOM is used to communicate with the EIOU and with the Computer Test Set (CTS).

The EIOU, whose design is based on Control Data's 480 computer which uses the Am 2901 microprocessor chip, consists of three modules: the General Processor Module (GPM), the Special Support Module (SSM), and the I/O Exchange Module (IOX). The GPM consists of data manipulation hardware, micromemory address control logic, and 1K of 48 bit word micromemory. The SSM contains 2K of 48 bit micromemory for use by the GPM, communication registers and multiplexers for communicating with the ECPU, event logic for timing and control, Real Time Clocks (RTC), Communications Security Interface Network (CSIN), and I/O memory access control. The RTC's, CSIN, and I/O memory access control perform the same functions as they do in the AN/GYK-12. The IOX module contains three TACFIRE type ac I/O channels, each capable of interfacing with eight devices, and a dc I/O interface which provides an interface between the EIOU and the ac I/O interface and provides for I/O expansion up to 15 external ac I/O channels.

Communication between the ECPU, EIOU, and the EMCMU's takes place over the external buses called bus 0 and bus 1. The external buses also interface the ECPU with the EIOU for communication with internal nonmemory devices, such as the RTC's, CSIN, and Diagnose Register. Bus 0 is accessed only by the ECPU for memory operations with the EMCMU. Bus 1 is shared by the ECPU and EIOU for both memory and nonmemory operations. The two buses have identical operating characteristics.

Two Bus Extender Modules (BEM's) increase the driving capabilities of - and provide receivers for - the external buses which interface the ECPU/EIOU to the EMCMU's. These modules are functionally transparent to the ECPU/EIOU and the EMCMU interfaces.

The Panel Interface Module (PIM) contains the registers and control logic to hold the information in the indicators and switches on the front panel and to communicate that information to and from the EIOU.

The two Power Supply Modules in the ECPU and the PSM in the EMCMU are identical. They convert 270 volts dc to +5 volts, +15 volts, and -12 volts.

Functionally the EMCMU is composed of Core Memory Modules (CMM), Bus Extender Modules, a Memory Control Module (MCM), and a Power Supply Module.

The eight CDC-14 CMM's each contain 32K by 18 bits of core which include a parity bit per 8 bit byte. This totals to 131K of 32 bit words of core memory per EMCMU, which is the same as the MCMU of the AN/GYK-12. Internal to the CMM's are an interface card and a storage assembly. The storage assembly contains the magnetic cores and drive sensing circuits.

The BEM's of the EMCMU are similar to the BEM's of the ECPU/EIOU except the former lack terminating resistors on the bus lines. The BEM's provide two ports for access to the EMCMU's.

The MCM contains the timing and control logic necessary to interface the memory to the two buses from the ECPU/EIOU.

Both the ECPU and EIOU have their own Computer Test Set (CTS) channel. The CTS serves as a maintenance console for the emulator and allows for program control and emulated instruction set modification through the use of the read/write micromemory in the CTS. Other features include the dynamic display of memory and registers on a CRT and the capability to modify them through keyboard action. Ability to set conditional transfer and halt switches and to set address, operand, and level breakpoints are additional features.

EMULATOR FAULT DETECTION AND ISOLATION

Since the emulator architecture differs from that of the AN/GYK-12, the software fault detection and isolation programs of TACFIRE will not isolate a fault to a card, even though they may indicate that there is a fault. Since one of the emulator development ground rules is that the existing TACFIRE software may not be changed, firmware fault detection is built into the emulator to isolate faults to the card. A two digit octal display isolates the faulty card and indicates its location by row and column in the case. The fault display may be advanced through four possible fault locations, with the most likely location being displayed first. The diagnostics are written such that after the first card has been tested, only previously tested circuits are used to test each card. If the diagnostics fail before the first card is checked, that card is most likely at fault.

ECPU/EIOU diagnostics and non-destructive memory diagnostics are run each time the emulator is powered up. ECPU/EIOU extensive memory diagnostics can be run by pressing the Fault Isolation Diagnostics (FID) button after switch selecting memory diagnostics.

EMULATOR EXPANSION CAPABILITIES

The AN/GYK-12 emulator is designed to allow expansion to meet the needs of foreseeable applications. The I/O section allows for expansion of up to 15 channels or 120 external devices. Up to 16 EMCMU's may be daisy-chained to the ECPU/EIOU. Two additional ports may be added to the EMCMU to obtain a four port memory. Instructions may be modified or added to enhance the instruction set of the emulator.

With these expansion capabilities, and especially with the instruction set flexibility provided by the micromemory of an emulator, the future for the AN/GYK-12 emulator and other emulators looks very bright.

REFERENCES

1. AN/GYK-12 Computer Principles of Operation Manual, Litton Systems, Jan. 72.

2. AN/GYK-12 Emulator for Battalion (LP) TACFIRE System 570043A Processor (CPU/IOU), Control Data Corporation, 1977.

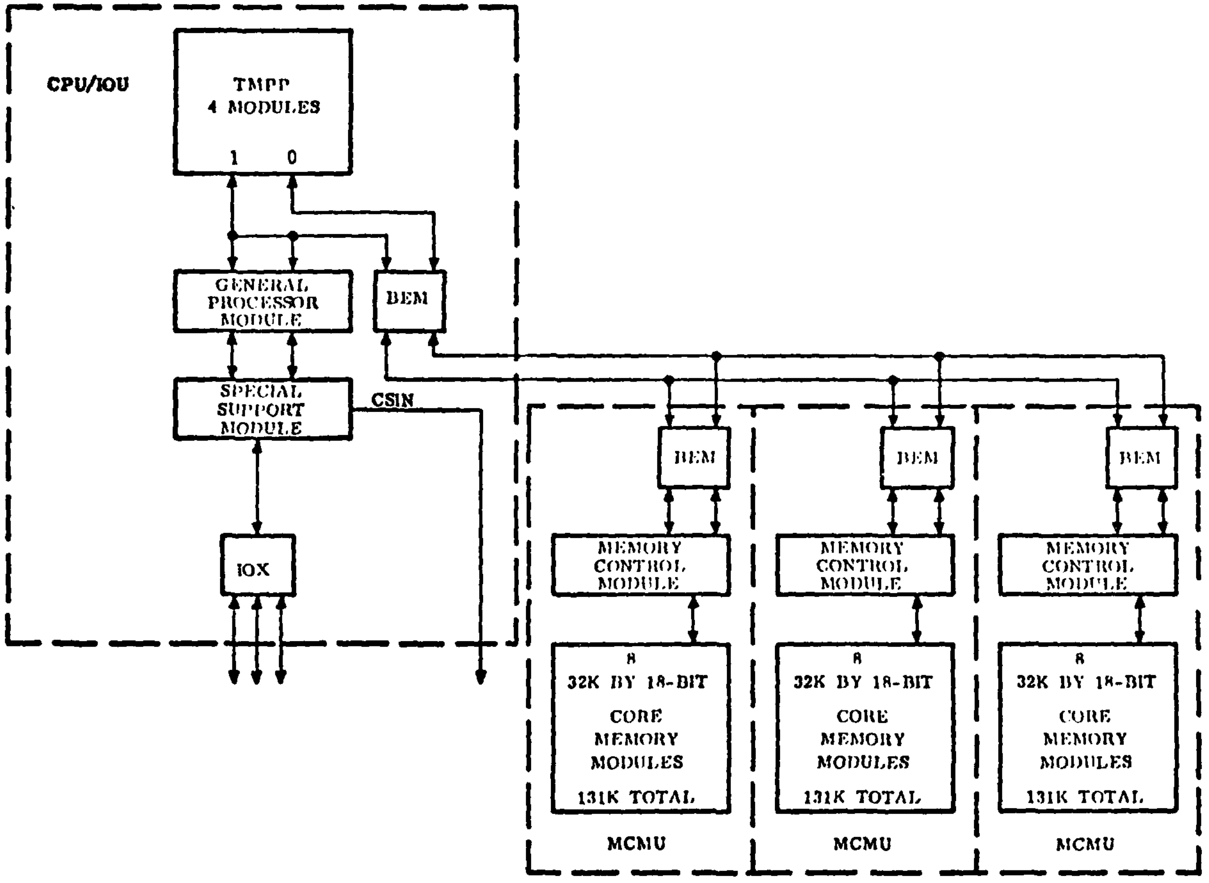
3. AN/GYK-12 Emulator for Battalion (LP) TACFIRE System 570044A Mass Core Memory Unit (MCMU), Control Data Corporation, 1977.

4. Beach, E. and Mercurio, J., "Emulation Capabilities of a Microprogrammable Multi-Processor System", Proceedings of the Seventh Annual Pittsburg Conference on Modeling and Simulation, April 1976, 9-13.

5. US Army Contract DAAB07-77-C-3033, "AN/GYK-12 Emulation", Fort Monmouth, NJ.

TACTICAL AN/GYK-12 EMULATOR

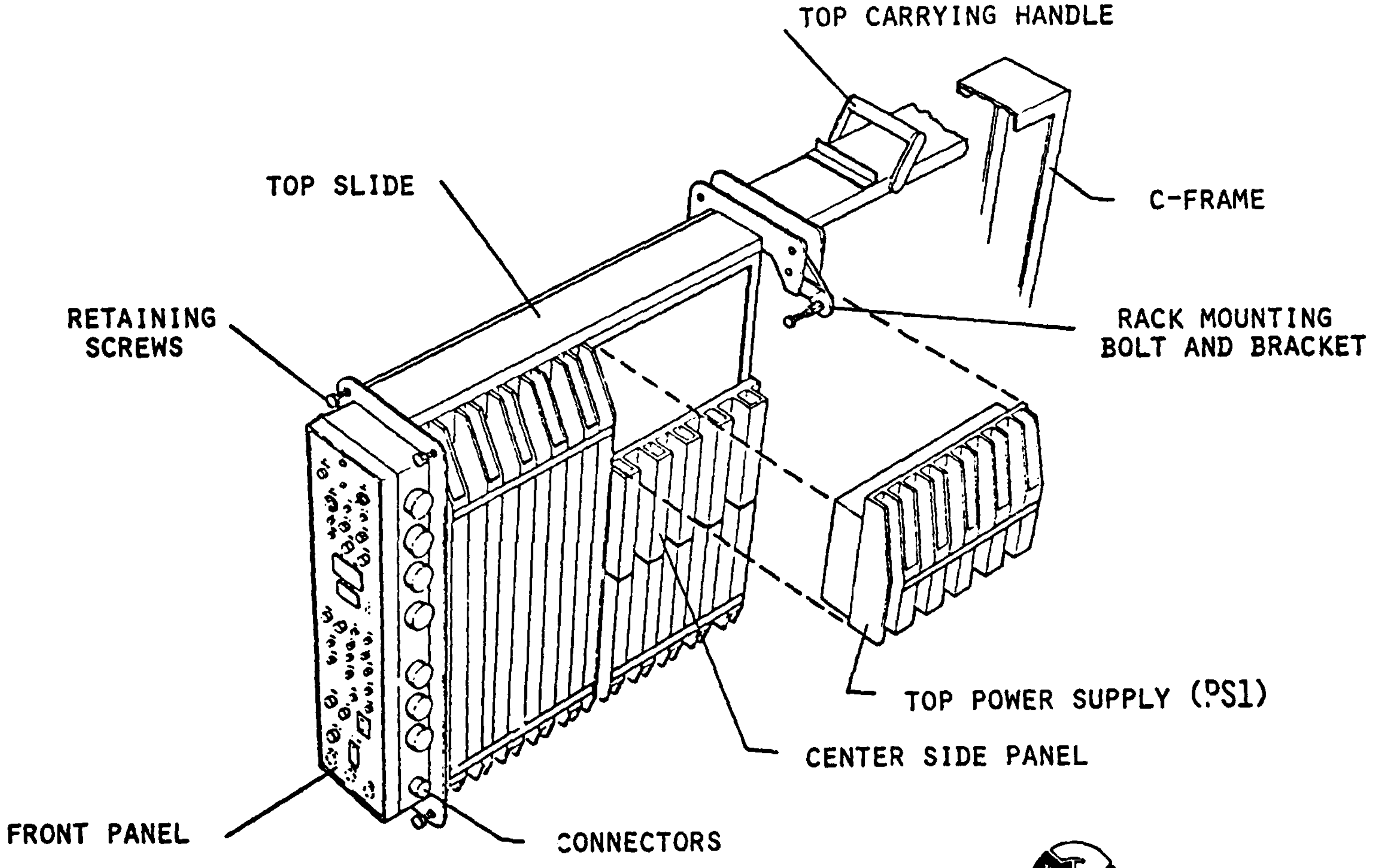
EMULATOR BLOCK DIAGRAM



CENTACS

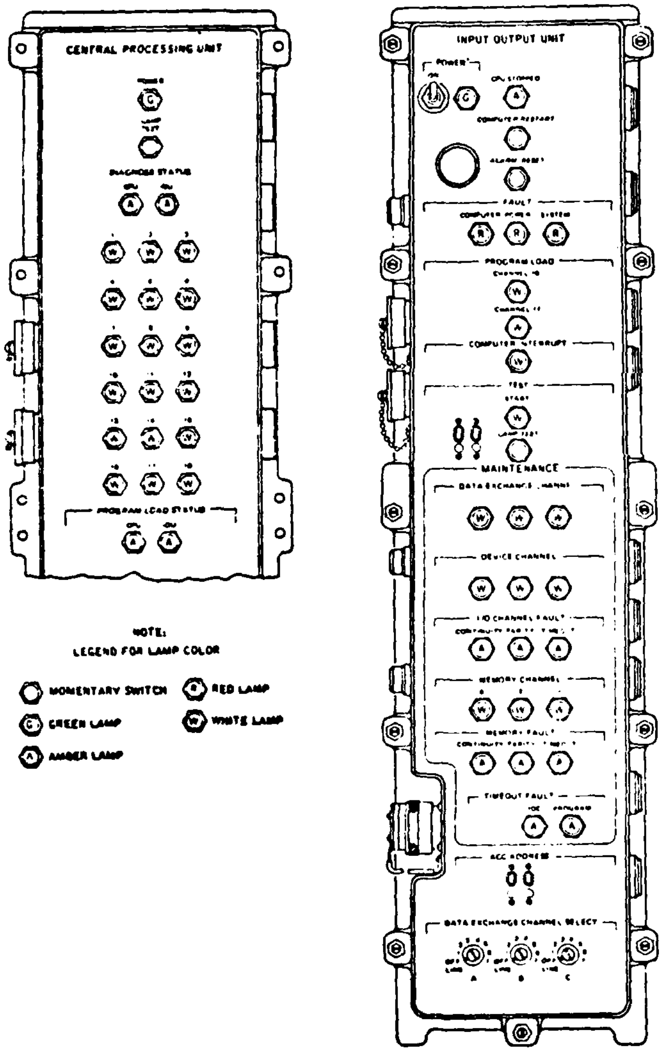
TACTICAL AN/GYK-12 EMULATOR

EMUI ATOR CPU/IOU CONFIGURATION



CENTACS

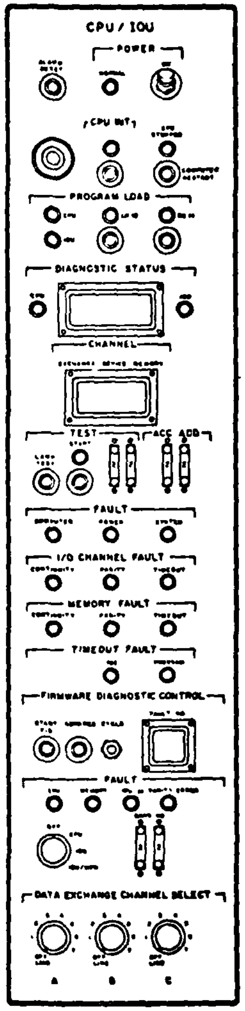
AN/GYK-12
CPU AND IOU FRONT PANEL



NOTE:
LEGEND FOR LAMP COLOR

- MOMENTARY SWITCH
- R RED LAMP
- G GREEN LAMP
- W WHITE LAMP
- A AMBER LAMP

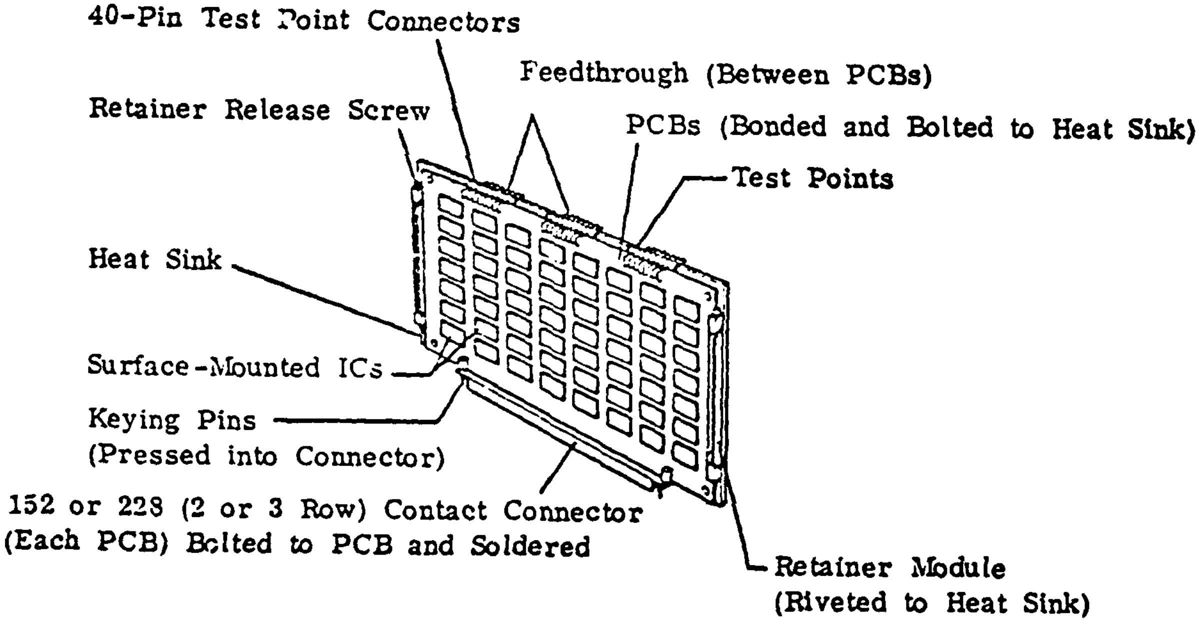
AN/GYK-12 EMULATOR
CPU/IOU FRONT PANEL



CENTACS

TACTICAL AN/GYK-12 EMULATOR

DOUBLE CIRCUIT CARD ASSEMBLY



40-Pin Test Point Connectors

Feedthrough (Between PCBs)

Retainer Release Screw

PCBs (Bonded and Bolted to Heat Sink)

Test Points

Heat Sink

Surface-Mounted ICs

Keying Pins
(Pressed into Connector)

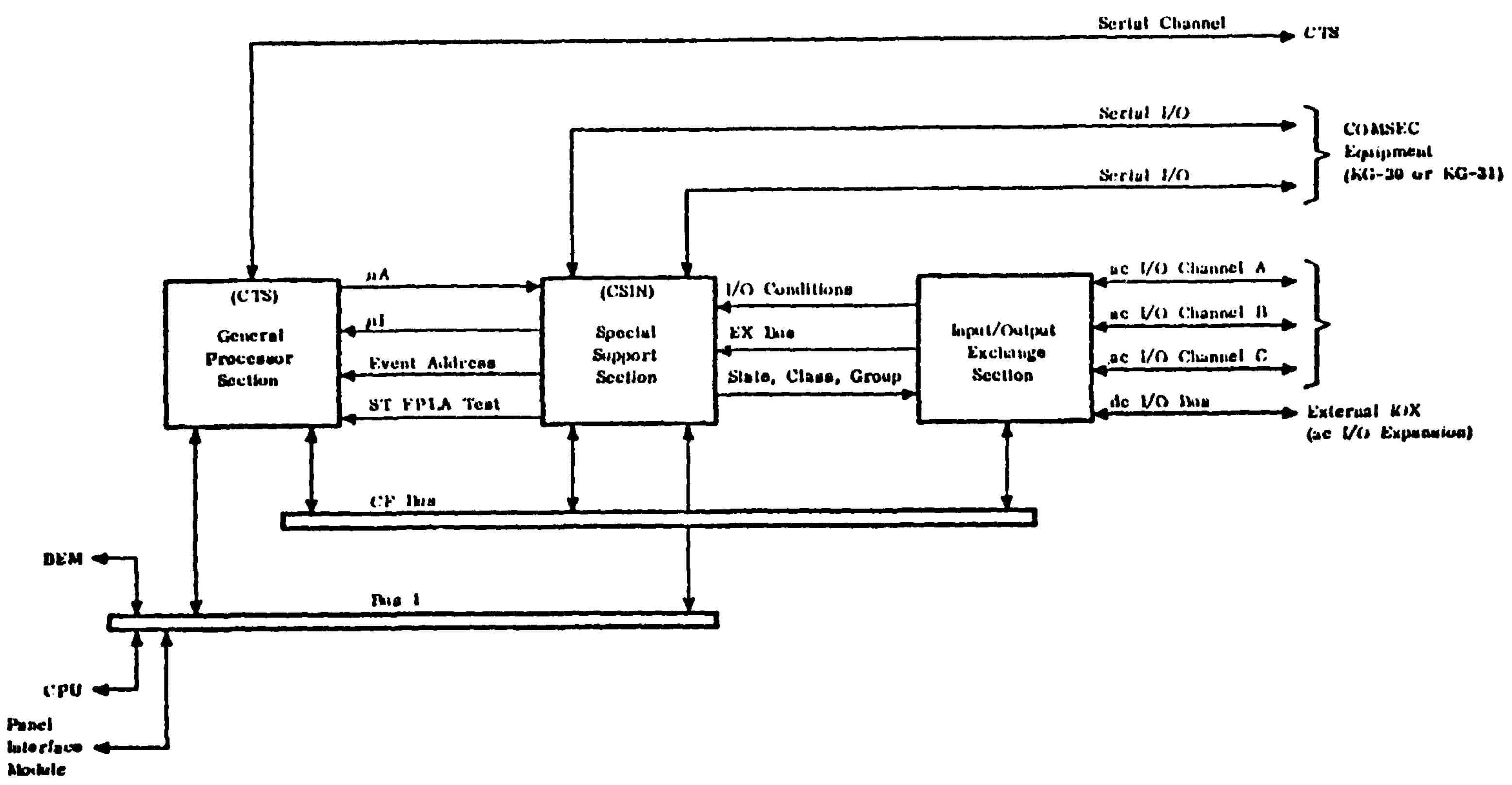
152 or 228 (2 or 3 Row) Contact Connector
(Each PCB) Bolted to PCB and Soldered

Retainer Module
(Riveted to Heat Sink)

CENTACS

TACTICAL AN/GYK-12 EMULATOR

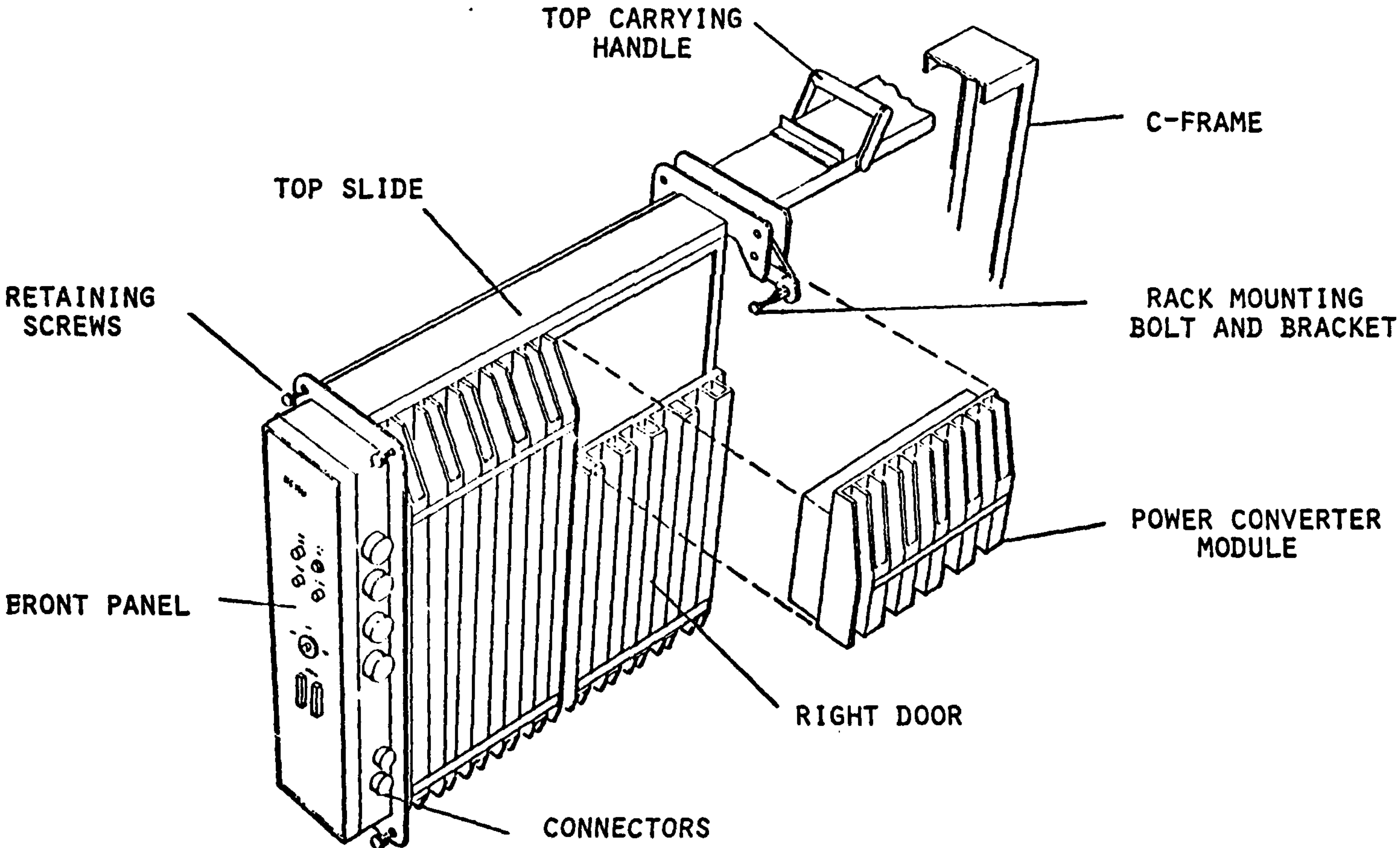
EMULATOR IOU BLOCK DIAGPAM



CENTACS

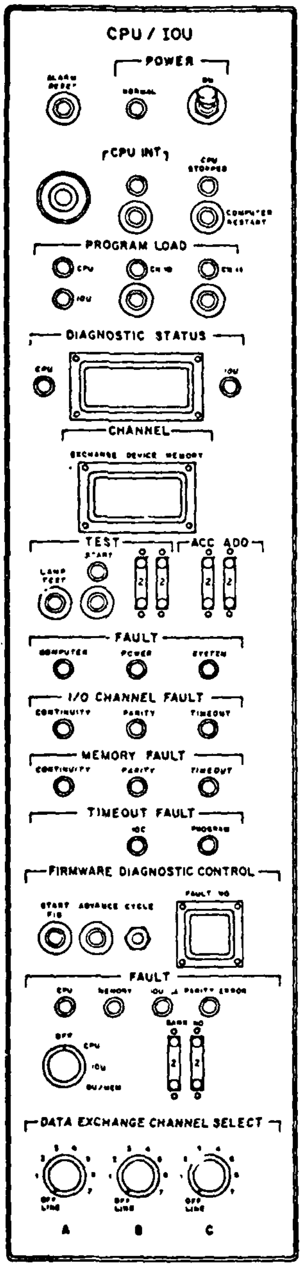
TACTICAL AN/GYK-12 EMULATOR

EMULATOR MCMU CONFIGURATION

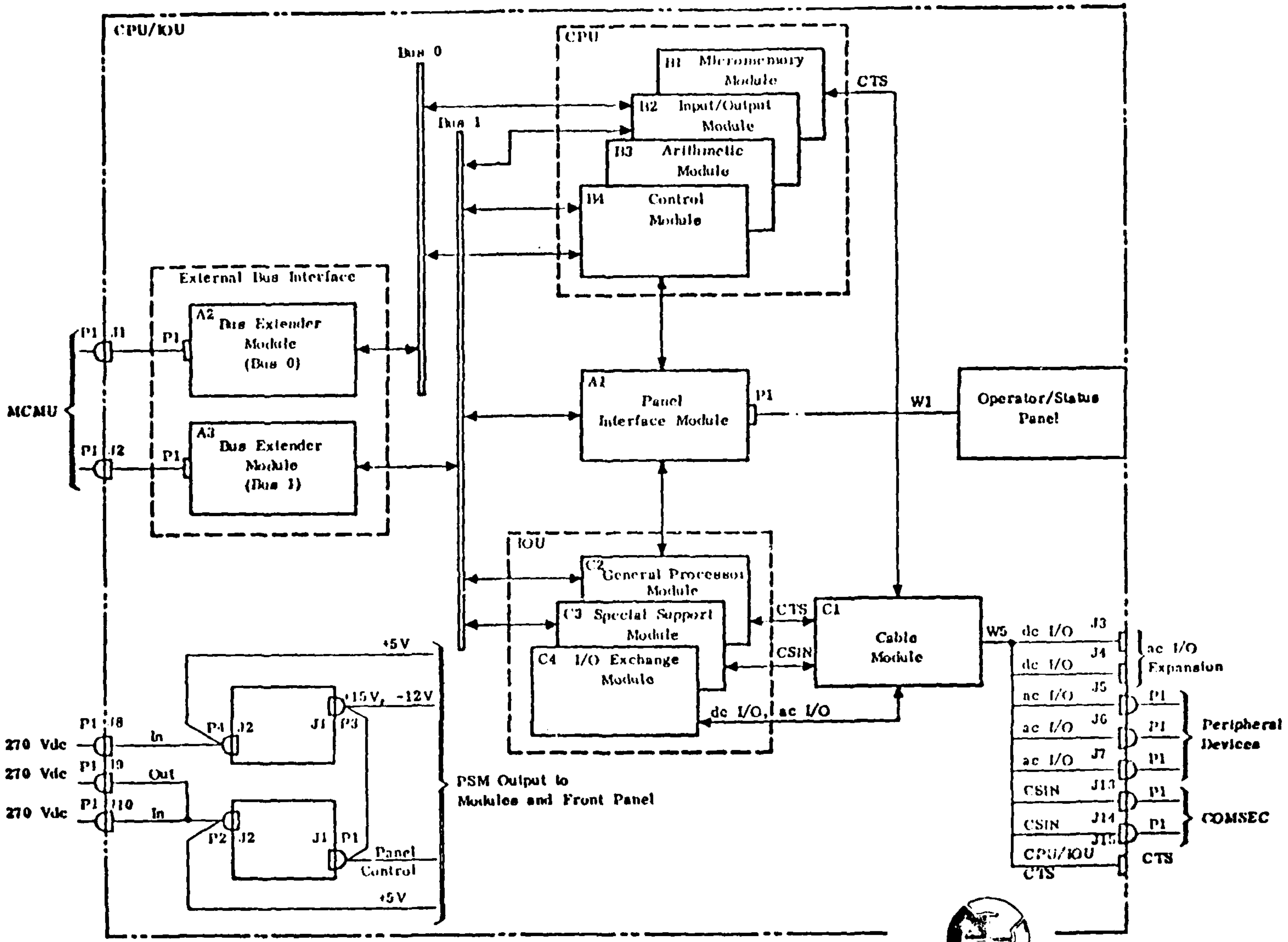


CENTACS

AN/GYK-12 EMULATOR
CPU/IOU

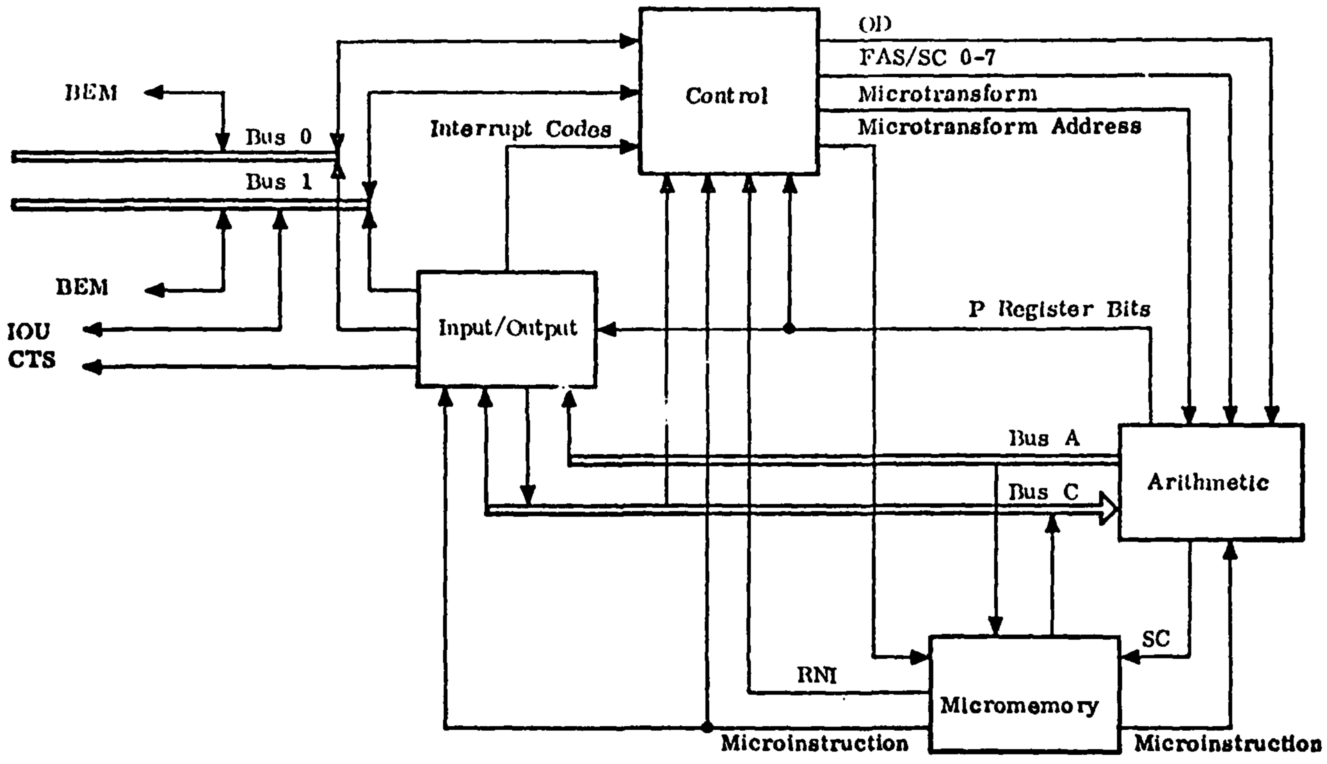


AN/GYK-12 EMULATOR CPU/IOU BLOCK DIAGRAM



TACTICAL AN/GYK-12 EMULATOR

EMULATOR CPU BLOCK DIAGRAM



CENTACS

TACTICAL AN/GYK-12 EMULATOR

COMPUTER TEST SET FEATURES

- READ / WRITE MEMORY
- DYNAMIC DISPLAY OF MEMORY AND REGISTERS
- MEMORY AND REGISTER MODIFICATION CAPABILITY
- CONDITIONAL TRANSFER AND HALF SWITCHES
- INSTRUCTION, OPERAND, AND LEVEL BREAKPOINT CAPABILITY

CENTACS

TACTICAL AN/GYK-12
EMULATOR

FIRMWARE DIAGNOSTIC FEATURES

- BUILT INTO EMULATOR
- ISOLATES FAULT TO A CARD
- FAULT INDICATION BY CARD ROW AND COLUMN
- CHECKS CARDS WITH PREVIOUSLY TESTED CIRCUITS
- NON-DESTRUCTIVE ECPU/EIOU AND EMCMU TESTS
- EXTENSIVE EMCMU DIAGNOSTICS

CENTACS

US ARMY CORADCOM

TACTICAL AN/GYK-12 EMULATOR

AN/GYK-12 CHARACTERISTICS

- 32-BIT INSTRUCTION WORD
- 1, 8, 16, 32, AND 64 BIT DATA WORD
- MEMORY EXPANSION TO 2M WORDS
- MEMORY ACCESS CONTROL AND PROTECTION
- 100 BASIC INSTRUCTIONS
- 9 ADDRESSING MODES
- SIXTEEN 32-BIT GENERAL PURPOSE REGISTERS
- 16 PAGE REGISTERS PER LEVEL
- I/O AT RATES UP TO 400,000 WORDS PER SECOND
- 64 PROGRAM LEVELS

CENTACS

TACTICAL AN/GYK-12 EMULATOR

EXPANSION CAPABILITIES

- UP TO 15 I/O CHANNELS
- UP TO 16 MCMU's
- UP TO 4 USERS PER MCMU
- SPECIAL INSTRUCTIONS

CENTACS

TACTICAL AN/GYK-12 EMULATOR

EMULATOR REQUIREMENTS

- INSTALL HARDWARE WITHOUT SHELTER MODIFICATION
- EXECUTE ALL TACFIRE SOFTWARE WITHOUT MODIFICATION
- MEET OR EXCEED AN/GYK-12 PERFORMANCE
- OBTAIN IDENTICAL LOGIC RESULTS
- DRIVE EXISTING TACFIRE PERIPHERALS
- PROVIDE EXPANSION CAPABILITIES

CENTACS

TACTICAL AN/GYK-12 EMULATOR

TACTICAL EMULATION ADVANTAGES

INJECTION OF TECHNOLOGY WITH COMPLETE RETENTION OF SOFTWARE (APPLICATION AND SUPPORT) WHICH LEADS TO:

- INCREASED RELIABILITY, AVAILABILITY, MAINTAINABILITY
- DECREASED COST, SIZE, WEIGHT, POWER
- INCREASED SPEED DUE TO NEW TECHNOLOGY
- PERMITS EXPANSION OF INSTRUCTION SET

CENTACS

Technology Upgrade of Existing System Peripherals

Jeffrey S. Yohay
Martin I. Wolfe

CENTACS

Technology insertion is the only cost-effective means of upgrading developed/fielded system hardware. However, technology insertion may necessitate redevelopment of system software, negating any performance benefits/ cost savings that might have accrued.

The use of smart peripherals driven by dedicated firmware provides a means for technology insertion without disruption of operational software. The smart peripheral contains its own embedded computer system to interface with mainframe hardware/software. This embedded system is driven by software programmed in firmware; application of the smart peripheral to an existing system without disrupting operational software is a matter of firmware redesign.

The Center for Tactical Computer Systems (CENTACS), Fort Monmouth, N.J., was able to employ the advantages of this technology to develop a replacement for the existing mass memory peripherals used in the Tactical Operations System (TOS) computer-based command and intelligence system. This replacement, a militarized disk mass memory, was programmed in firmware to emulate the existing drum memory devices, and provided PM TOS the opportunity to gain considerable cost/space savings in the fielded systems.

TECHNOLOGY UPGRADE OF EXISTING SYSTEM PERIPHERALS

Jeffrey S. Yohay and Martin I. Wolfe

US Army Communications Research
and Development Command
Center for Tactical Computer Systems
Software Engineering Division
Fort Monmouth, New Jersey

1. Introduction

High technology computer-based systems are employed by the government for many diverse applications. In non-defense applications they may be large information and record keeping systems; in the military, they may be command and intelligence aids, or may control a great variety of electronic hardware. All of these systems have one thing in common: they are all products of commercial computer technology, and all will inevitably be obsoleted by that same technology.

In this paper, the Army's Center for Tactical Computer Systems (CENTACS) Fort Monmouth, NJ would like to present its "lessons learned" from a recent project undertaken to upgrade an Army computer-based system. These "lessons" have provided many insights into the avoidance of that inevitable obsolescence mentioned above, and have provided new strategies to effect the technological upgrade of an existing system at minimum cost and with maximum flexibility.

2. DOD Computer-Based Systems

A computer-based system is one centered about a central processing unit (CPU), an input/output unit (IOU), and associated main memory and peripherals. Included among the peripherals are auxiliary mass memory devices (magnetic disks, drums, tapes), and input/output (I/O) devices (CRT and keyboard terminals, line printers, and card readers and punches). These devices are controlled by main-memory-resident software, typically in the form of a CPU-dependent operating system (OS). This OS controls, through the IOU, those peripheral devices required during applications software execution on the CPU. The entire computer-based system may be "embedded", i.e., a one-application dedicated system running within a hardware system and performing a series of tasks for the operation of that hardware only. In this case it may or may not have an OS; the single application program may control peripheral devices and perform its own I/O. Additionally, the program may be contained in "firmware", i.e., hardware memory devices programmed through hardware alteration ("burning-in" of bits in a semiconductor memory, for instance) to perform a software function.

DOD computer-based systems are typically closed application systems designed to perform a well defined number of tasks. These systems usually have strict performance, environmental and stress requirements which are rarely faced by commercial computer systems.

For these reasons, the DOD system is usually designed "ground up" to meet the needs of a specific user. Improvement or revision of such a system becomes limited by those same requirements and the initial design. The existing systems must advance to the new design goal through the bounds set by the user. These bounds may be application oriented or financial; nevertheless, their imposition often results in abandonment of planned system upgrades. Eventually, the only means of reaching the new design goals is through another "ground-up" system design - a wasteful and inefficient process.

3. Technology Insertion

The CPU/IOU and its memory/peripherals are products of one of the United States' fastest-growing high technology industries. CPU/IOU architecture has grown tremendously versatile, allowing multiple user and remote timesharing applications, and providing vastly decreased hardware execution time. Physically, a system comprising of a CPU, IOU, and 64K (1K = 1,024) words of memory that required an entire room 15 years ago can now be placed on a few semiconductor chips. Semi-conductor memory devices have gone from 1K to 4K to 16K to 64K bits per chip in the space of a few years.

Along with this fast-growing technology has come dramatic increases in hardware performance, and co-responding decreases in hardware cost. In the Army's own Tactical Operations System (TOS), the old-technology magnetic drum mass memory device is capable of storing 9.3 million bits, at a cost of 0.8 cents/bit; a modern-technology magnetic disk mass memory device can store 663.7 million bits, at a cost of 0.1 cents/bit! Technological advances have extended the limits of hardware performance to areas undreamt of by early system designers, and with less cost and less physical space requirements.

Thus, the problem faced by computer-based system managers is one of "technology insertion": the insertion of high technology devices into an existing system to improve system performance and lower system cost. This problem exists because of the modular nature of these systems; technological advances may occur in one or more of the hardware types, but not in others. Order-of-magnitude (factor of 10) performance increases or cost decreases may occur in any one hardware area, justifying a system upgrade in that area alone. Additionally, overall system budget constraints that would otherwise prevent continued system improvement will have less effect if system planners can upgrade each hardware area a lone. The original system design objective can be retained while system hardware is upgraded.

4. The Software Problem

Dramatic improvements in hardware technology are not, unfortunately, the only yardstick by which to measure computer-based system performance. Despite order-of-magnitude advances in hardware, systems are still controlled by software, and software remains the biggest stumbling block to overall system improvement. Improved software management techniques and block-oriented high-order languages can assist in reducing software costs, but nothing has advanced software technology to the level achieved by hardware. Whether it requires 1 minute or 1 hour to execute a software algorithm, the greatest cost remains in the many hours required to formulate, write, and debug it by an applications programmer, and the many more hours to be spent by the same or other programmers to upgrade and maintain it.

Thus, it is in the interest of system planners to minimize disruption to operational software. The amount expended in software upgrades can be considerable; add to that the cost of maintaining a new software package, and the life-cycle cost can be quite high indeed. The worst case would be the expenditure of that money without gaining either a significant increase in system performance (through new features, or improved algorithms for old ones), or a significant decrease in system cost (through language improvements or improved programming techniques for simpler maintainability).

Technology insertion that disrupts operational software results in that very same worst case. Software must be changed (not upgraded) to accomodate the new device; no software performance improvement is required or expected. A tremendous expenditure of limited funds must be made in software to effect an improvement in hardware. As a result, the system planner is usually forced to forego the technology upgrade; the software cost increase negates the hardware cost decrease (or performance increase), and the system is forced to remain at a static level of technology. The result is obsolescence, and eventual forced replacement by another system built "ground up" with modern technology, only to begin the wasteful cycle anew.

5. The Smart Peripheral Solution

Technology insertion without disruption of system software can be effected only through use of upgraded hardware that can be operated with existing software. At first, that would seem to be a simple solution; the problem lies in the great diversity of software and hardware, requiring a new "solution" every time an upgrade is made to the existing system. Peripherals would have to be designed specifically for each system to be upgraded. The design (or re-design) of peripherals for individual systems could be done only at great expense in time and money. Moreover, many government applications require highly specialized equipment, and generally of a much greater ruggedness than that required by commercial environments. The result is often a long, ground-up design cycle that begins with modern technology, but results in and obsolete device (as compared to present technology) when finally fielded. The technology thus inserted would already be obsolete!

Design of hardware to meet software requirements also imposes a tremendous area of specialty on what may already be a highly specialized piece of equipment. The additional cost of re-designing the hardware provides the mirror image of the software worst case: the hardware must be changed, not upgraded, to meet software operational requirements. Again, the tremendous added expense many negate the expected system improvements, and the new technology will not be utilized.

The recent upgrade to the TOS system by CENTACS faced the problem of technology insertion without software disruption, and found what may be a most desirable solution. TOS is a command and intelligence system, receiving information on enemy troop movements from forward observers, and forming a large data base from which commanders and intelligence officers may draw needed information to aid in making battlefield decisions. The computer used is the Army's AN/GYK-12 CPU with its associated IOU and peripherals. Among those peripherals are magnetic drum and tape mass memory devices, upon which the intelligence data base is to be placed.

The drums are to provide fast, on-line access to the data base; the tapes are used to store ("dump") information as necessary, and to save any desired data that can be stored off-line. Both are specialized, militarized units drawn from existing (1960's) computer technology. Both employ obsolete technology, out-dated by the lengthy ground-up design time required to field them.

The drums provide the worst example of this hardware obsolescence. They consist of a controller and up to 8 drums, occupying almost half of a standard Army S-280 truck-mounted shelter(!) Their total possible capacity is under 75 million bits, at a total (1977) cost of over $600,000. This compares to commercial magnetic disk memory units storing up to 660 million bits, and costing $40,000, that can fit a fraction of the required space! Worse, the drums are inadequate for their task; TOS Operable Segment (TOS^2) field tests resulted in several complete system shut-downs due to insufficient drum storage and several "throttles" when messages were temporarily not accepted for the same reason. This test was run with a 4-drum system; purchasing 4 more drums at a cost of $285,000 provides a most limited solution.

The tape units, though not as much affected by obsolescence as the drums (far less development has taken place in this area), have the same problems as with all tape storage systems; extreme slowness of data access and easily damaged tapes. Data can be accessed over 10,000 times faster on the drum than on the tape, and field conditions play havoc with the tapes and their associated hardware. Their unreliability make their replacement as an effective mass memory device imperative.

The problem, then, is to insert modern mass memory commercial technology into the TOS system, and provide significant performance increase and life-cycle cost decrease while doing so. The latter constraint makes it imperative that TOS software operate unchanged; but how can this be accomplished?

The great diversity of hardware and software mentioned earlier provides both the problem and the solution: Commercial peripheral manufacturers, faced with this same diversity, turned to the "smart peripheral": a peripheral device with its own embedded computer system performing the simple application of controlling the peripheral in response to CPU/IOU commands. This embedded system is run by a driver program stored in firmware, which can be programmed to drive the device with different command sets, depending on the system into which the peripheral is placed. By redesigning the driver firmware, one peripheral can serve the needs of a great variety of users.

Thus, the smart peripheral provides the solution to technology insertion without software disturbance. Though the aim of the insertion is to provide the system with the high-technology "dumb" end of the device (i.e., the mass memory), the "smart" end of the peripheral allows the mainframe hardware/software to talk to this "dumb" end and control it for proper system utilization. Most importantly, ground-up peripheral design is no longer necessary for individual system upgrades; successful technology insertion efforts can now be measured in months, not years, and at far less expense.

The availability of militarized smart peripheral devices enabled CENTACS to investigate the applicability of this technology for replacement of the TOS drums. The result was the use of a Control Data MD640 militarized disk unit, programmed in firmware to emulate the drum units, with the potential of replacing the magnetic tapes at any desired time in the future. Two of these units used as disks (not as emulators) could provide order-of-magnitude increases in storage capacity, and would cost $480,000 less per TOS system than the present drum and tape devices. The resultant savings in physical space will allow TOS to place their present two-shelter system into one S-280 shelter, saving both a shelter and an M-36A five ton truck to transport it; the life-cycle cost savings could amount to over 50 million dollars!

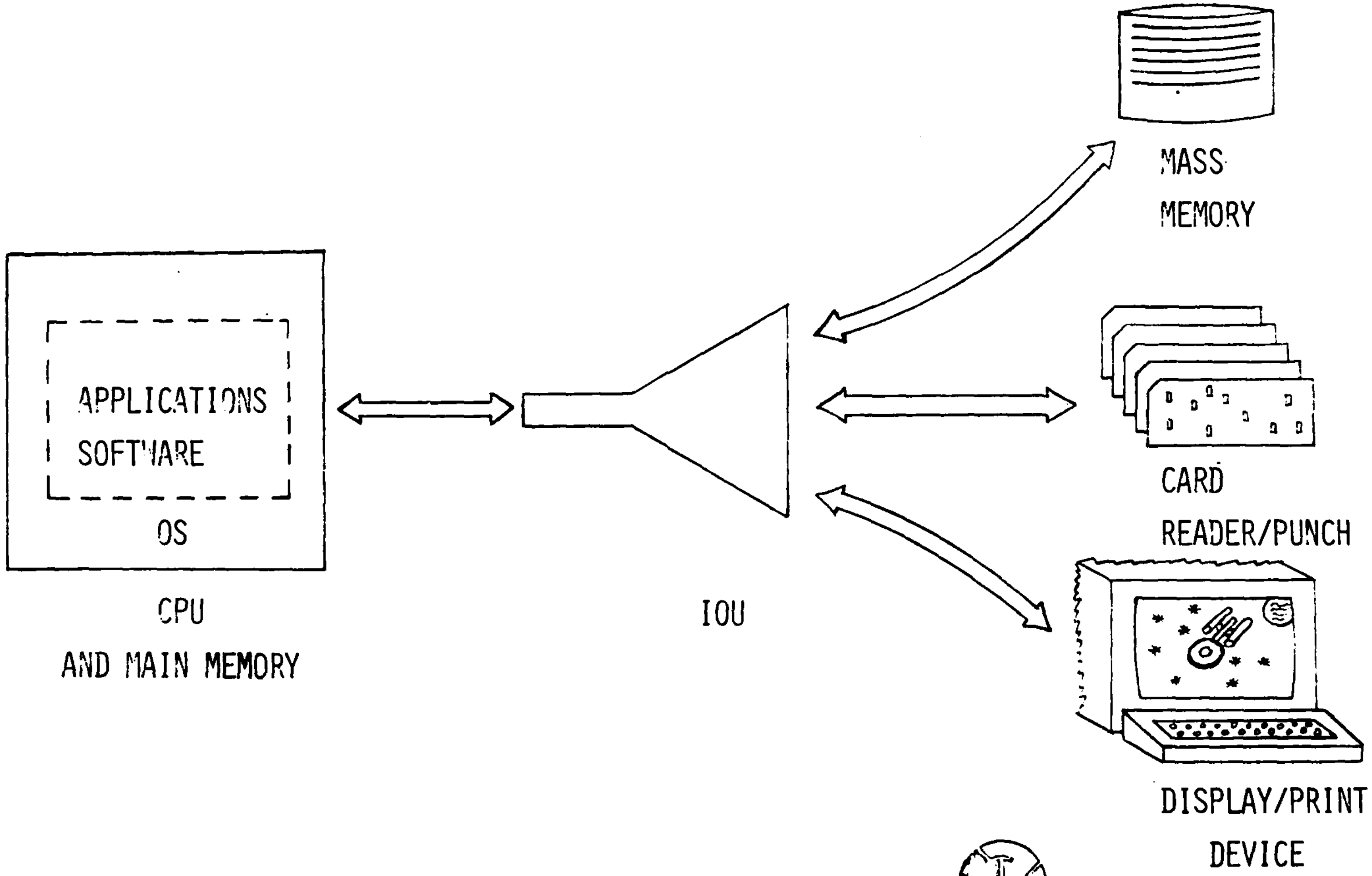
It may also be noted that future TOS upgrades, whether in hardware or software, that affect operation of the drum emulator disk, can be included into the device through firmware re-programming. The smart peripheral thus guards against its premature replacement due to system upgrades in other areas.

6. Summary

In summary, it is clear that important ground has been broken in the development of government computer-based systems. The need for re-design of entire systems to keep pace with advancing commercial technology can be eliminated by the use of technology insertion, while technology insertion is made economically feasible by the use of the smart peripheral.

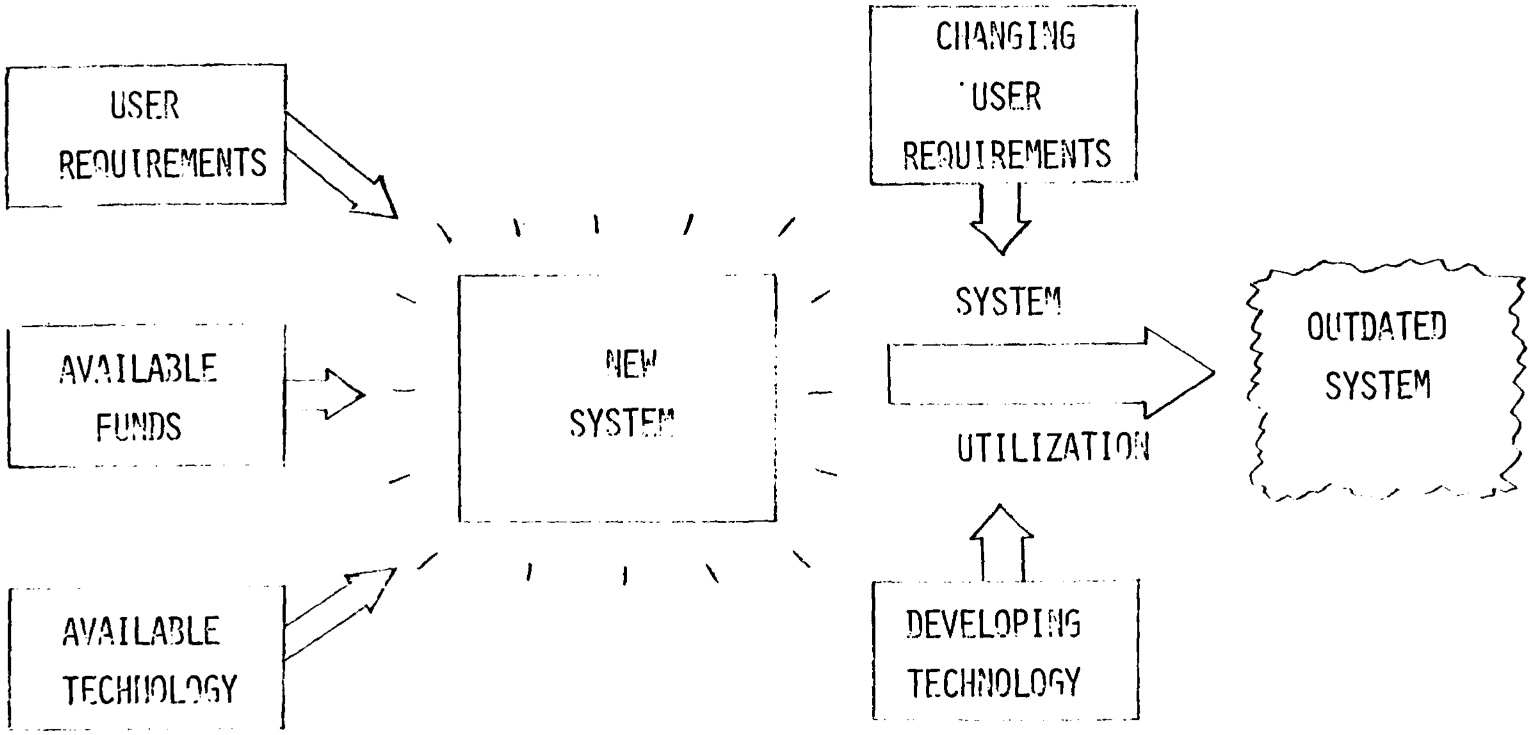
TECHNOLOGY UPGRADE
OF EXISTING SYSTEM PERIPHERALS

COMPUTER-BASED SYSTEM



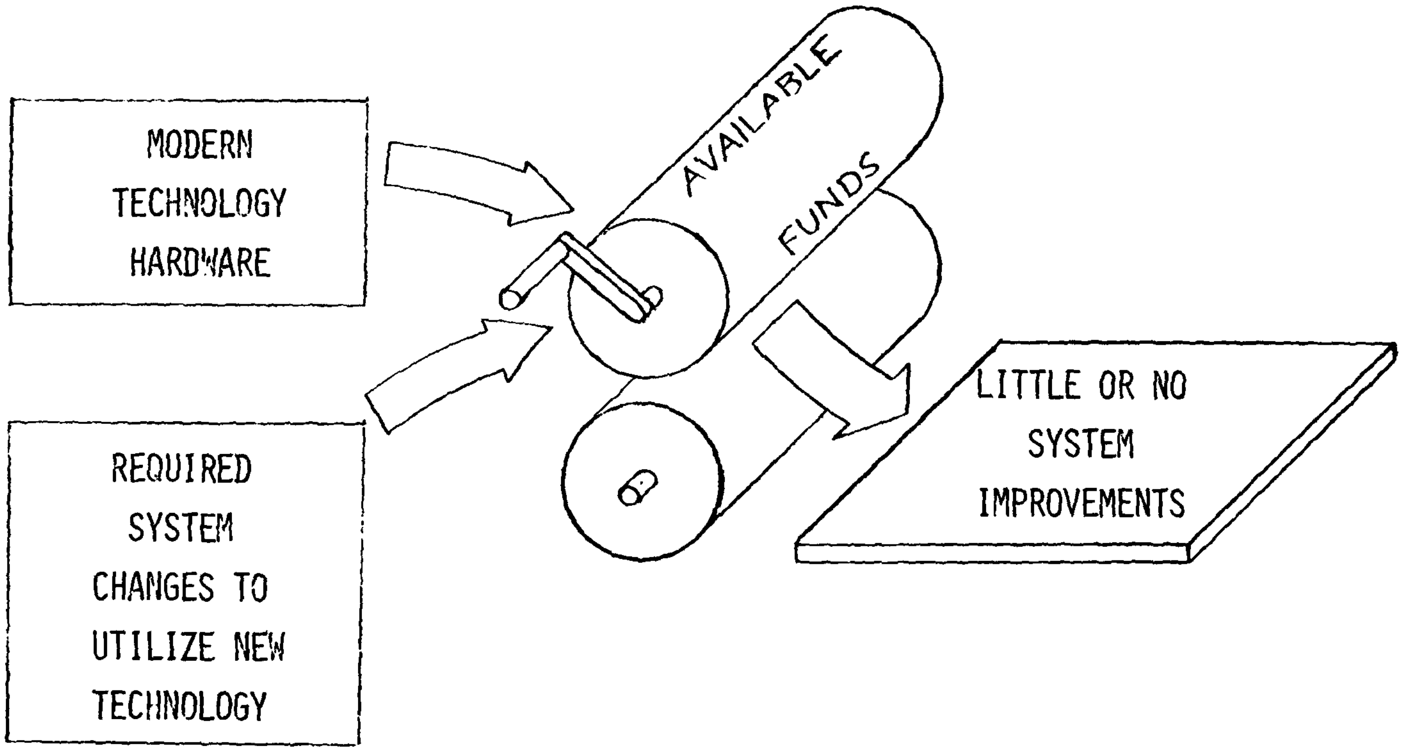
TECHNOLOGY UPGRADE
OF EXISTING SYSTEM PERIPHERALS

SYSTEM DESIGN CYCLE



TECHNOLOGY UPGRADE
OF EXISTING SYSTEM PERIPHERALS

THE ECONOMIC WRINGER



MODERN TECHNOLOGY HARDWARE

REQUIRED SYSTEM CHANGES TO UTILIZE NEW TECHNOLOGY

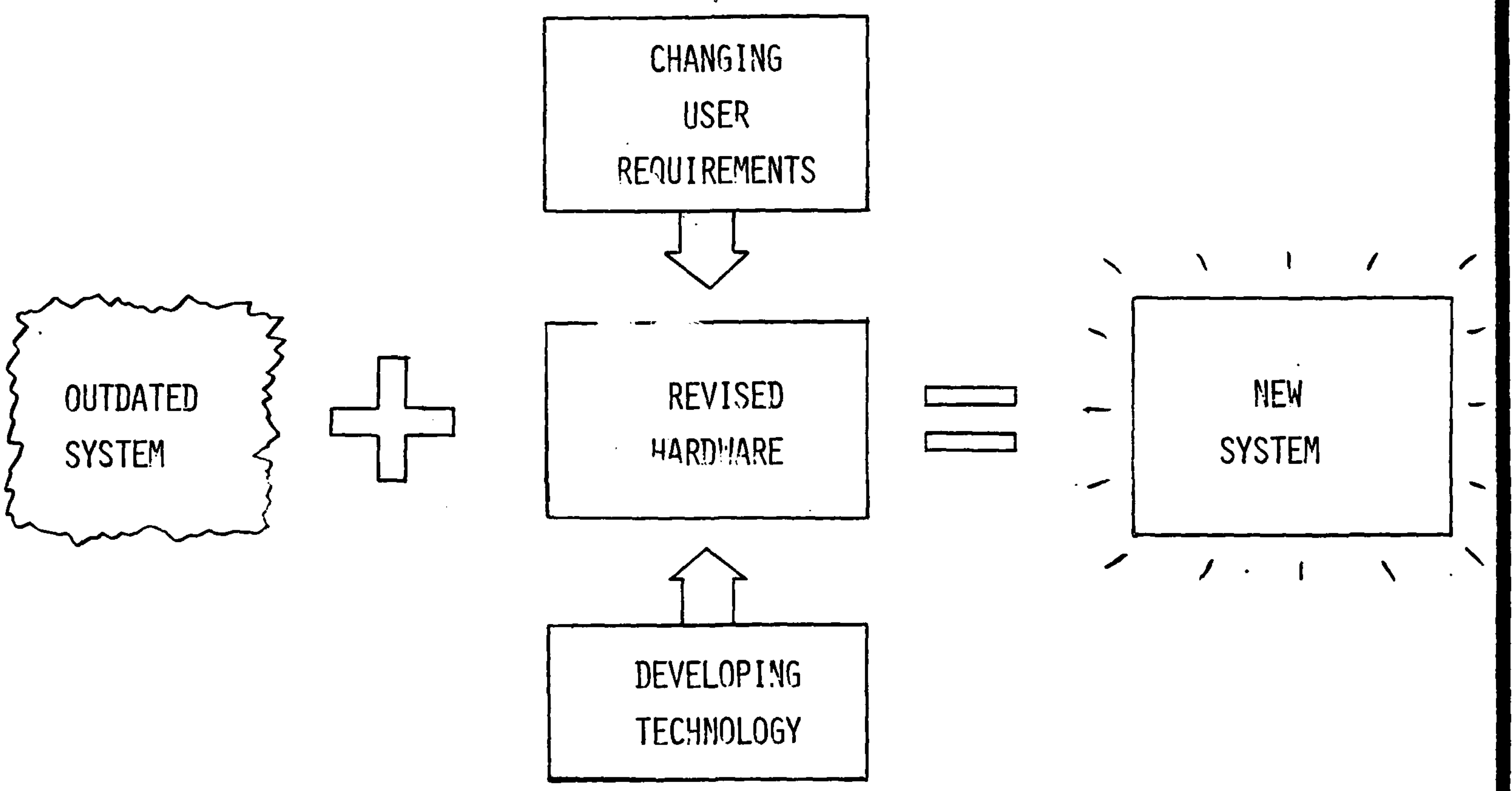
AVAILABLE FUNDS

LITTLE OR NO SYSTEM IMPROVEMENTS

CENTACS

TECHNOLOGY UPGRADE
OF EXISTING SYSTEM PERIPHERALS

SYSTEM UPGRADES



CENTACS

A Case Study of the Software/Firmware Development for a Microprocessor-Based Computer

James E. Scott

U.S. Army Missile Research & Development Command
Missile System Software Center, Redstone Arsenal

This paper described the software/firmware development approach taken on a small experimental missile computer with no available support software. A cross assembler was written which allowed programmer definition of micro/macro assembly language and a first attempt was made at microcoding for a digital autopilot application. It was soon obvious that some higher level programming approach would be needed to cope with impending program modifications. At a minimum, it was judged that a macro instruction set should be defined to simplify the programming process; however, severe time and memory loading requirements appeared to dictate optimization to the micro level.

The solution chosen was to develop a compiler from a specialized, limited HOL to a "trial" macro instruction set. Sample runs were made and analysis of the compiler output resulted in the design of a macro instruction set highly optimal for the application. An editor/optimizer was then written to incorporate the new instruction set as the last stage of compilation. The finished product has proven to be flexible and highly effective. Also, because the scope of the development was limited, only a low-level effort by personnel having little translator writing experience was required over a short period of time.

A CASE STUDY OF THE SOFTWARE/FIRMWARE DEVELOPMENT FOR A MICROPROCESSOR-BASED COMPUTER

James E. Scott

Missile System Software Center
Guidance and Control Directorate
US Army Missile Research and Development Command
Redstone Arsenal, Alabama

I. INTRODUCTION/BACKGROUND

This paper outlines the history and technical approach taken on the software/firmware development for a microprocessor application. This work was performed by the Missile System Software Center, US Army Missile Research and Development Command, over the period 1975-1978. The application is a digital autopilot (DAP) for a 6 inch diameter missile.[1,2] The DAP computer is an experimental, in-house design based on the first commercially available 4-bit slice bipolar microprocessor chip (Monolithic Memories 67Ø1). The computer is 16 bit and contains 1024 words of main memory (256 RAM, 768 ROM) and 1024 X 50 bit words of microprogram memory (ROM). Microinstruction execution time is 750 nanoseconds and size is 6 inch diameter X 3.5 inches with power requirement of 50 watts max. The objective of the hardware design was to demonstrate that a small, minimal hardware configuration digital computer could be built with potentially low production costs and could serve as a modular design for multiple missile applications.

Given the minimal hardware approach, it was expected that the DAP application would load the computer significantly. The design approach of a completely microprogrammable machine was believed to be a means of providing enough processing power from a small amount of hardware to solve the real-time flight problem. From a software development point of view, the need to do microprogramming on a "minimal hardware" machine had the appearance of a difficult hand optimization task. It certainly would have seemed incredible to consider that a higher order language (HOL) with all its advantages and "inherent inefficiencies" might be used on such a project. However, as this paper will show, an HOL approach was chosen that proved to be extremely effective at maximizing the performance of a minimal hardware machine.

II. APPROACH

II.1. Initial Support

The first software support efforts were an emulator and a controller/debug program. The Emulator[3] was developed concurrently with hardware design and has proved to be a powerful tool for studying basic characteristics of the machine design and for identifying specific features which might require special or unusual attention by a programmer. In order to check out the first lab prototype computer, a test facility was required. The Controller/Debug System[4] was developed based on an available minicomputer facility (see Figure 1). This controller/debug system was designed and software was written concurrently with latter stages of design of the DAP computer. From the system teletype an operator can load programs in octal, run programs, inspect memory and internal registers, insert break points, and perform various other functions and tests.

II.2. The Assembler

Entry of programs of more than a few words in length from a teletype is, of course, tedious, time consuming, and error prone. An assembler was definitely needed. However, at the early stage of program development no macro instruction set had been defined; and even though the emulator had provided microcoding experience, a micro instruction set had not yet been adequately identified. A suitable solution was a user-definable (meta) assembler.

A cross assembler was developed on another minicomputer facility with disc (see Figure 1). This assembler was designed to process both micro and macro code. It is interesting to note features of the assembler which permitted a very quick implementation.

1) Very rigid "fixed format" source statements - Fixed format simplified the task of scanning source statements. Although free field is not available, the user can specify his own format.

2) User defined symbols (mnemonics) - This feature, along with the fixed format, actually made the assembler designer's job easier since continuing support for an expanding source language was put in the hands of the user.

3) The assembler is coded in structured FORTRAN. Translator writing (or any program coding) in structured HOL is, of its nature, a big plus.

The conception, design, and coding of the first version assembler was accomplished in one man month. This work included an object tape output module and a loader linked with the controller/debug system. An additional two man months has been expended on the assembler for the addition of features (e.g. symbol printout table) and the accommodation of an expansion of the microprogram memory word size from 48 to 50 bits.

II.3. Initial Application

With an assembler available one analyst spent six months designing, microprogramming, and testing a first version of the DAP application software. The coding was done from a "microcoder's" point of view: the program was viewed as "micro" in nature with the main program being resident in microprogram memory and tables of data and special address links being resident in main memory. When major changes in the DAP flight equations were considered, the estimate for a microcoded software redesign was close to the same six man months which had been required for the original equations of similar difficulty but different form. Also, it was expected that the DAP equations might be modified several times before a version was accepted for flight tests; and follow-on programs might involve further modifications. It was obvious that some other coding scheme would be desirable in order to expedite the programming task.

II.4. The Macro/Micro Dilemma

A clearly simple approach to making the DAP computer programming task reasonable would be to drop microcoding, define a macro instruction set, implement the macro instruction set in microcode, and do coding essentially in "normal" assembly language. However, analysis of the program already written in microcode and evaluation of even more demanding new requirements indicated that memory and time loading were already problems. It was difficult to envision a partitioning between main and microprogram memory which would be much better than the very efficient one which had been developed for the first microprogram. The two analysts reviewing the problem were not aware of any approach for proving that what had been done was nearly optimal! Several arrangements of main and microprogram memory were studied, but no significant improvements were discovered and an adequate macro instruction set was not obvious.

The problem stated in more general terms:

> Given a specific application for a microprogrammable computer, how can one develop software and firmware to produce a highly optimal utilization of the hardware?

It is interesting that the solution chosen for this application was sparked by departing from the "micro view" at the machine level and by looking instead to the system requirements. From the requirements it was apparent that equation solving was the largest part of the DAP processing, that sequence control was rather trivial, and that I/O was relatively straightforward and could be handled efficiently by special purpose macros. Indeed, there was no reason that the requirements could not relatively easily be translated into some HOL (the application had been previously coded in FORTRAN for off-line simulation purposes). If an appropriate HOL program could be used to state a solution to the processing problem in a rather efficient manner, such an HOL program could, in effect, be used interchangeably with the requirements statement. The problem noted above might then be restated:

> Given an HOL program, how can one efficiently translate it to a microprogrammable computer? Specifically, how can macro/micro tradeoffs be made for a more effective hardware utilization?

Of course, what is asked for above is an optimizing compiler. But this compiler would have to resolve the macro/micro tradeoff dilemma. Such a resolution did not appear trivial. Compiler optimization for any target machine is not a particularly easy task, and microprogramming is an added dimension of flexibility and difficulty. The partitioning of macro and micro code is really an allocation of resources problem. How many macro's of what size are needed? What combinations will produce good or even adequate results? A compiler capable of such a level of resource allocation might take considerable development effort (a luxury not affordable on this R&D project).

II.5. The Compiler

A compiler usually translates an HOL program into one or more intermediate forms; an intermediate language (IL) output can be the means for making a compiler more readily retargetable. The IL which might be output by a compiler can be very target machine independent and, for certain classes of translation (i.e. computation), can be an efficient and straightforward representation of the source HOL program. In other words, the DAP equations written in HOL might be translated into an IL (possibly a simple assembly language) which would represent those requirements (equations) about as efficiently and completely as the HOL statements. This implies that if a proper IL could be chosen as output of the compiler, the IL translation of the HOL program could be an accurate and reasonably efficient mapping of the requirements. If this IL were chosen such that it could be reasonably implemented as a macro instruction set on the DAP computer, a compiler could solve the processing problem from a functional point of view but not necessarily from an efficiency standpoint. A question arose: Why not choose an IL, compile to it, examine the output code and generate from that a more appropriate (time/space efficient) macro set?

Such work appeared feasible, but the time frame and level of effort required was of prime importance. As was noted earlier, this particular application is computation intensive. Greater than 90% of time and memory is delegated to the solution of arithmetic equations. This fact was the key to a means of implementing an HOL approach on a low budget. Why implement a full HOL like FORTRAN when a simple subset would suffice? Also, since the project was essentially experimental in nature, what justification could there be in attempting to "generalize" the approach? Why spend many man years trying to solve the world's problems when a few man months of effort could produce an effective solution for the specific application and at the same time could maintain enough flexibility to handle a class of similar problems?

The compiler with editor/optimizer was developed in four man months. It is coded in structured FORTRAN and resident on the program generation minicomputer system shown in Figure 1. The following discussion presents some information pertinent to the compiler effort.

The HOL chosen is a very specialized FORTRAN subset plus data declarations. See Figure 2 for a list of features. The need for data declarations can be appreciated by realizing that the hardware support is for only 16 bit integer add/subtract, that multiply is microcoded, and that no other computational operations are available. Divide was not required for the application; therefore, a divide macro was not written. With only integer operations available, scaling had to be handled in some fixed point fashion. At the beginning it was not exactly clear what fixed point arrangements would be feasible; full data declarations facilitated the modification/addition of data types in the compiler.

Sequence control involved only some simple decision making and did not appear to warrant any particular degree of sophistication. Therefore, the FORTRAN arithmetic IF was chosen for its ease of implementation. The IF is limited to test of single data items (not expressions). Recent efforts have resulted in updating sequence control to structured forms. This was done by utilizing an available preprocessor and limiting logical tests to comparison with Ø (equivalent to the arithmetic IF test). See Figure 2 for a listing of the structured forms employed.

The restriction on the IF test is a rather crude limitation, but it is indicative of a "get the job done" attitude. It is also indicative of a rather inexperienced compiler writer, a point worthy of note! The author had very little experience at compiler design, but was able to accomplish a successful "subset" HOL implementation.

The IL selected as output of the compiler is based on one accumulator and two levels of addressing. Only 256 words of main memory are directly addressable; indirect to one level is available for addressing RAM and 512 words of ROM. Figure 3 lists the compiler generated macro set (IL). It was anticipated that the "one accumulator" virtual machine would not make efficient use of the actual hardware which has 16 registers. However, compilation to a single accumulator appeared to be a worthy simplification with allocation of other registers possible in a post-translation effort.

Once the compiler with IL code generator had been completed, the DAP equations were compiled to the simple IL. Manual review of the generated IL led to the following observations:

1) Certain repetitive patterns of IL output suggested the possibility of combining those sequences into new macros. This was the important key to the development of more efficient macros.

2) For this application, only one additional register was required for arithmetic temporary results. New macros could implement this feature.

3) As was expected, some simple optimizations which had not been included in the compiler should be incorporated (e.g. elimination of NOP's and unneeded accumulator loads, reduction in the conditional branches generated for IF's).

A new set of expanded macros was designed and microcoded. An editor/optimizer was written to perform some simple optimization and edit in the new macros. This editor/optimizer was incorporated as the last stage of compilation. See Figure 3 for a listing of the editor/optimizer generated macros. The process of macro review and design was an iterative one which was considered complete after the most obviously useful new macros had been incorporated. The search for better macros was by no means exhaustive.

Figure 4 depicts a sample program segment translation from structured source to macro code. It should be noted that the listings are only partial; storage reserved for data and linkage and the actual microcoded macros are not shown.

III. RESULTS

Three versions of the DAP application have been coded:

1) Microcoded - This first attempt was only partially completed due to updates in requirements, changes in the computer design, and emphasis on the HOL approach.

2) Flight Version - This HOL version is the final result of numerous updates, including a state variable redesign, to the DAP equations; it has been tested in an in-the-loop simulation and has been burned into PROMS for the first flight test.

3) Structured Flight Version - This is a redo of the flight version in structured form. It was not available in time to meet testing requirements for the first flight test. Subsequent DAP software will be developed using structured forms. For this application structured forms had no adverse effect on efficiency.

Unfortunately, an accurate quantitative comparison between the microcoded (1) and HOL (2 or 3) versions has not been made. One reason for this is that the HOL capability was available early enough to make updating of the microcoded version to the new state variable design unnecessary. Also, although an HOL version of the microcoded version is possible, no such effort has been attempted. The analyst responsible for all application software (except the original microcoded version) and for development of the DAP macro sets has intimate familiarity with both the micro and HOL approach. It is his evaluation that the efficiency of the compiler generated code is comparable to that of the microcoded version.

The use of HOL has greatly reduced turn-around time for application program changes. This reduced turn-around time has enabled system designers to view the software as less fixed and more experimental, a great advantage in a research program.

IV. CONCLUSIONS

1. The HOL approach developed was effective on a "minimal hardware" computer. Subsetting and specializing of the HOL made possible rapid development of the compiling system by personnel having little translator writing experience. The availability of a facility for developing translators in structured HOL was beneficial.

The HOL effort expended on this rather small scale project is further evidence that machine level coding, even for time-critical application software, should be a dying art.

2. Firmware, when combined with the compilation process, proved to be particularly effective at maximizing the performance of the minimal hardware computer. It is noted that the expanded macro set is the product of hand optimization based on recognition of repetitive sequences of IL output; thus, the compiler is capable of compiling efficiently to only a specialized class of equations. As a matter of expanding compilation to capitalize on microcode capability for a larger class of problems, it would be feasible to develop a macro generator which could automatically generate a set of appropriate macros for each compilation.

REFERENCES

1. Asquith, C. F., T-6 Digital Autopilot Data Processing Analysis and Specification, Report No. RG-75-36, US Army Missile Research and Development Command, Redstone Arsenal, Alabama, March 1975.

2. Plunkett, K. W. et al, Design and Analysis of a Microprocessor-Based Digital Autopilot for Terminal Homing Missiles, Report No. T-78-57, US Army Missile Research and Development Command, Redstone Arsenal, Alabama, March 1978.

3. Brookshire, J. R., Multipurpose Digital Microprocessor Emulator, Report No. RG-76-62, US Army Missile Research and Development Command, Redstone Arsenal, Alabama, May 1976.

4. Baxter, W. F., Microprocessor Controller/Debug System, Report No. RG-76-61, US Army Missile Research and Development Command, Redstone Arsenal, Alabama, May 1976.

MICROPROCESSOR DEVELOPMENT FACILITY

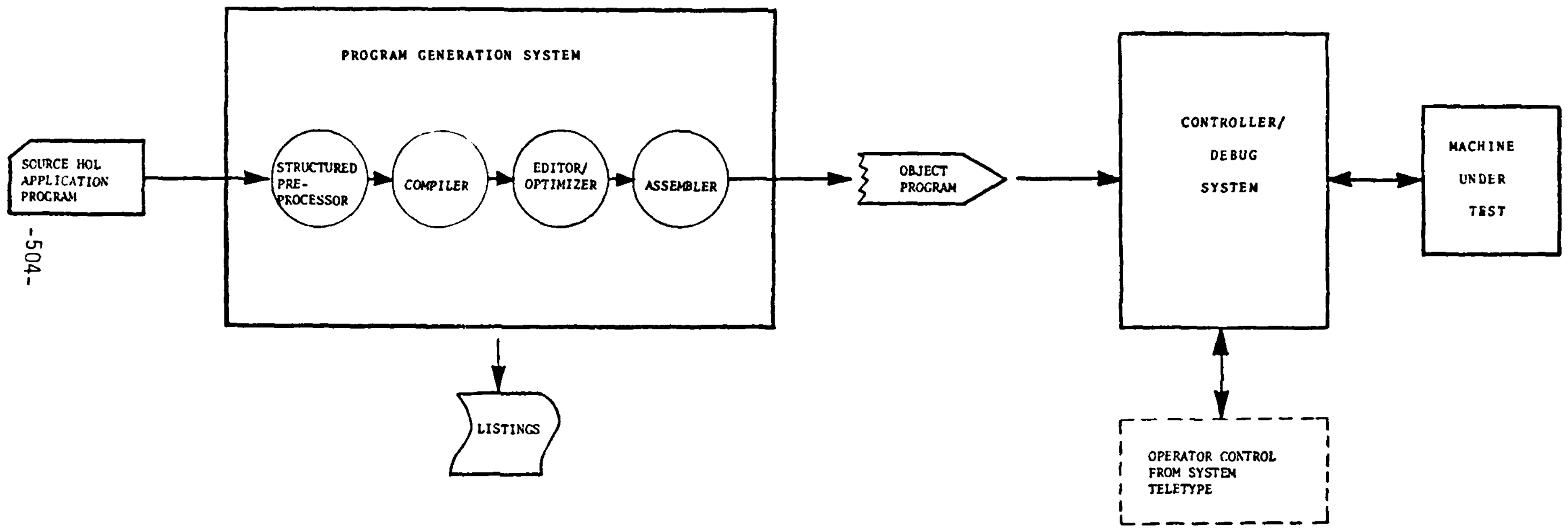


FIGURE 1

DAP HOL FEATURES

The HOL is subset FORTRAN with full data declarations required and extended to structured sequence control forms by means of a preprocessor.

Data Declarations

DECLARE ident:	VARIABLE		(fixed point only)
	CONSTANT	X.XXX	(decimal)
	INTEGER	XXX	(decimal)
	OCTAL	XXX	
	ADDRESS	ident	(used to build or access data structures)
	LINK	ident	
	INTERNAL	XXX	(compiler information)

Computation

Standard FORTRAN arithmetic replacement statement

ident = arithmetic expression

Sequence Control

IF(*)-THEN-ELSE-ENDIF
IF(*)-THEN-ORIF-THEN-.....ENDIF
DO WHILE(*)-ENDDO
DO UNTIL(*)-ENDDO

*The only test allowed is one variable compared with Ø.

Arithmetic IF
GO TO

(processed by the compiler as output by the preprocessor; programmer use is discouraged)

CALL

(presently used only for direct macro code insertion)

Other

Comments (C in column 1)
CONTINUE
END
Special direct micro or macro code insertion

The following are compiler processed, but not implemented in the code generator:

READ, WRITE
SUBROUTINE
RETURN

FIGURE 2

DAP MACRO INSTRUCTIONS[1]

COMPILER GENERATED			EDITOR/OPTIMIZER GENERATED		
OPERATION	MNEMONIC[2]	EFFECT[3]	OPERATION	MNEMONIC	EFFECT[4]
Load	LD, LDI	M → A1	Save & Load	SAL, SALI	A1 → X1, then M → A1
Store	ST, STI	A1 → M	Store Zero	STZI	Ø → M
Add	AD, ADI	A1 + M → A1	Add to Memory	ADMI	A1 + M → A1 → M
Subtract	SB, SBI	A1 - M → A1	Add Temporary	AST, ASTI	A1 + M → A1 → X1
	MSB, MSBI	M - A1 → A1		ATSI	A1 + X1 → A1 → M
Multiply	MPS, MPSI	A1 x M → A1		ADAT	A1 + X1 → A1 → X1
(round)	MPR, MPRI	A1 x M → A1		TADA	A1 + X1 → A1
Scale	SCL, SCLI	Scale A1 by M	Subtract Temp	TSBA	X1 - A1 → A1
No-op	NOP		Branch (on -)	JPN, JPNI	Test A1
Branch	JP, JPI	Unconditional	(on ≠ Ø)	JPNZ, JPNZI	Test A1
(on Ø)	JPZ, JPZI	Test A1	(on +, ≠ Ø)	JPTP, JPTPI	Test A1
(on +, Ø)	JPP, JPPI	Test A1	(on Ø, -)	JPZN, JPZNI	Test A1

NOTE:

1. Several macro instructions are not shown in this table. These are instructions which are not generated by the compiler or editor/optimizer and must be coded thru CALL's or direct code insertion. The instructions are primarily I/O with some miscellaneous functions.

2. Mnemonics ending with the letter "I" indicate indirect addressing to one level.

3. M stands for the effective memory operand; A1 is the main accumulator. Branches are to the effective operand address.

4. X1 is an accumulator reserved for storage of temporary arithmetic results.

FIGURE 3

DAP PROGRAM SEGMENT PARTIAL LISTINGS

Source Program →
(Structured Pre-
processor Listing)

```
DO-NOTHING PROGRAM SEGMENT
    (DATA DECLARATIONS ARE NOT SHOWN)

IF(FLAG1.EQ.0)
   THEN
      X12=A08*BPHI-B08 *ROL - X11
      X11=A09*BPHI+B09*ROL
   ELSE
      IF(SWCH1.LT.0)
         THEN
            BPHI=BPHI+BPHIC
      ENDIF
      FLAG1=0
ENDIF
CALL EXEC1(EX1P)
END
```

Preprocessor Output →
(Compiler Listing)

```
C.     IF(FLAG1.EQ.0)
       IF(FLAG1)9999,8999,9999
8999   CONTINUE
C.     THEN
       X12=A08*BPHI-B08 *ROL - X11
       X11=A09*BPHI+B09*ROL
C.     ELSE
       GO TO  9998
 9999 CONTINUE
C.     IF(SWCH1.LT.0)
       IF(SWCH1)8997,9997,9997
8997   CONTINUE
C.     THEN
       BPHI=BPHI+BPHIC
C.     ENDIF
 9997 CONTINUE
       FLAG1=0
C.     ENDIF
 9998 CONTINUE
       CALL EXEC1(EX1P)
       END
       END$
```

Compiler Output →

```
*        IF(FLAG1)9999,8999,9999
       LDI        FLAG1
       JPZ        #8999
       JPP        #9999
       JP         #9999
* 8999  CONTINUE
#8999 NOP
*        X12=A08*BPHI-B08 *ROL - X11
       LDI        BPHI
       MPS        A08
       SCL        SAN0
       STI        TEMP1
       LDI        ROL
       MPR        B08
       MSBI       TEMP1
       SBI        X11
       STI        X12
*        X11=A09*BPHI+B09*ROL
       LDI        BPHI
       MPR        A09
       STI        TEMP1
       LDI        ROL
       MPR        B09
       ADI        TEMP1
       STI        X11
*        GO TO  9998
       JP         #9998
*  9999 CONTINUE
#9999 NOP
*        IF(SWCH1)8997,9997,9997
       LDI        SWCH1
       JPZ        #9997
       JPP        #9997
       JP         #8997
* 8997  CONTINUE
#8997 NOP
*        BPHI=BPHI+BPHIC
       LDI        BPHIC
       ADI        BPHI
       STI        BPHI
*  9997 CONTINUE
#9997 NOP
*        FLAG1=0
       LDI        0
       STI        FLAG1
*  9998 CONTINUE
#9998 NOP
*        CALL EXEC1(EX1P)
       EXEC1      EX1P
*        END
```

Edited/Optimized →
Compiler Output
(Assembler Listing)

Assembled
Macro Code

```
         IF(FLAG1)9999,8999,9999
      LDI        FLAG1                       0075 202764
      JPNZ       #9999                       0076 010356
* 8999  CONTINUE
*        X12=A08*BPHI-B08 *ROL - X11
#8999 LDI        BPHI                        0077 022763
      MPS        A08                         0100 010773
      SCL        SAN0                        0101 032374
      SALI       ROL                         0102 030765
      MPR        B08                         0103 015776
      TSBA                                   0104 027420
      SBI        X11                         0105 004756
      STI        X12                         0106 001767
         X11=A09*BPHI+B09*ROL
      LDI        BPHI                        0107 202763
      MPR        A09                         0110 015775
      SALI       ROL                         0111 030765
      MPR        B09                         0112 015777
      ATSI       X11                         0113 046366
*        GO TO  9998
      JP         #9998                       0114 206357
*  9999 CONTINUE
*        IF(SWCH1)8997,9997,9997
#9999 LDI        SWCH1                       0115 200772
      JPP        #9997                       0116 012360
* 8997  CONTINUE
*        BPHI=BPHI+BPHIC
#8997 LDI        BPHIC                       0117 002771
      ADMI       BPHI                        0120 044763
*  9997 CONTINUE
*        FLAG1=0
#9997 STZI       FLAG1                       0121 002764
*  9998 CONTINUE
*        CALL EXEC1(EX1P)
#9998 EXEC1      EX1P                        0122 035772
*        END
```

FIGURE 4

A CASE STUDY OF THE SOFTWARE / FIRMWARE DEVELOPMENT FOR A MICROPROCESSOR-BASED COMPUTER

THE COMPUTER

"Minimal Hardware" Design

Microprogrammable-

16 bit configuration
based on 4-bit slice
TTL chip

Memory-

1024x16 bit Main

1024x50 bit Microprogram

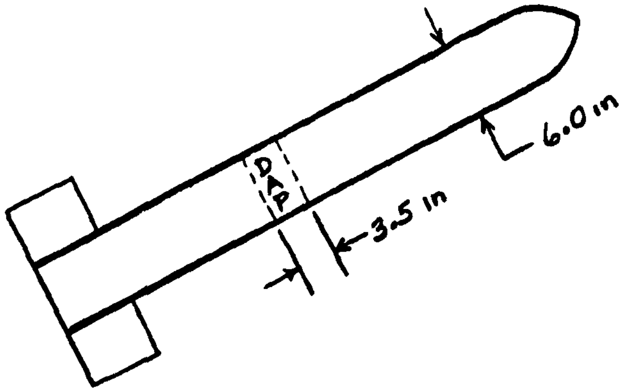
750 ns Micro Instruction Time

Power-

5v, 50 w max

THE APPLICATION

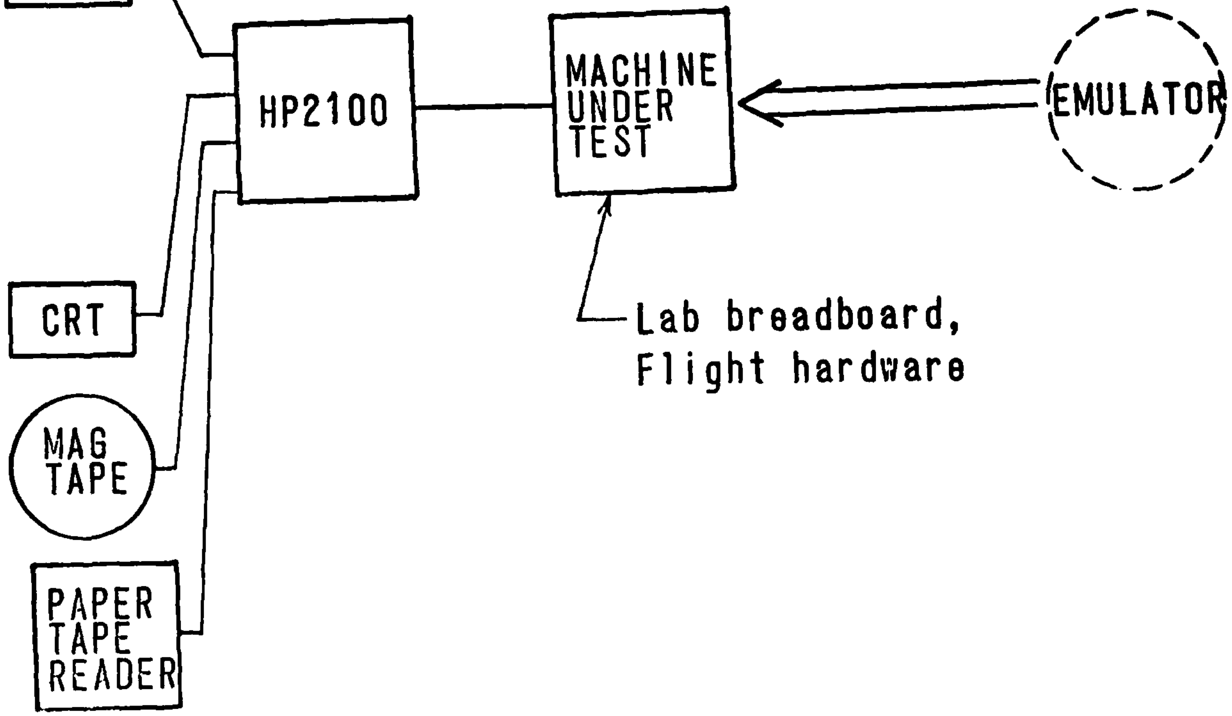
Digital Autopilot (DAP)



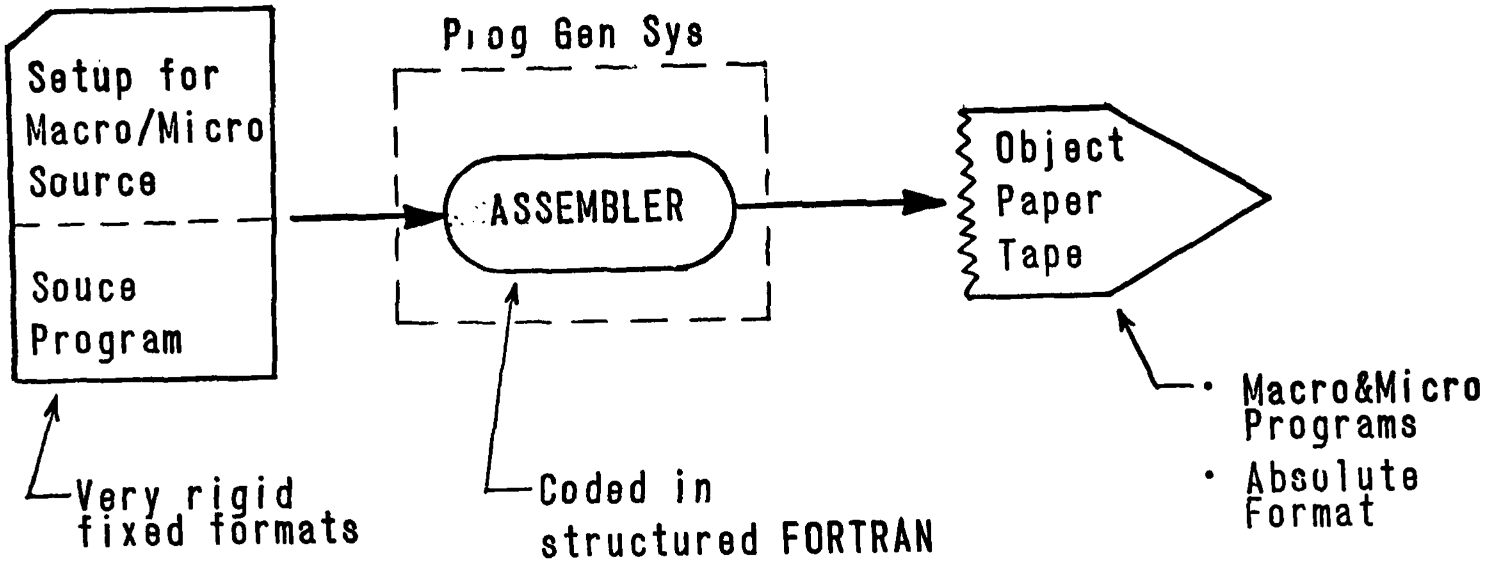
SPECIAL SUPPORT

CONTROLLER/DEBUG SYSTEM

- Control
- Load/mod/inspect memory& registers
- Breakpoint
- Tests, etc.
- Display data
- Record data



USER DEFINABLE ASSEMBLER



COMPOSITE
LISTING
FOR
PARTIAL
SET-UP
DECK
REQUIRED
BY
ASSEMBLER

```
**** DIGITAL AUTOPILOT ASSEMBLY - PASS 2 ****

*
*************************************************
*                                               *
*    *** DIGITAL AUTOPILOT MNEMONIC SET-UP ***  *
*                                               *
*************************************************
$COL
        1Ø  77   Ø   1              $EQU
        11   2   3   4              PC      ØØ   2
        15   5   6   7              R1      Ø1   2
        17  77  77  1Ø              R2      Ø2   2
        18  11  12  13              R3      Ø3   2
        2Ø  77  77  14              X1      Ø4   2
        21  15  16  17              X2      Ø5   2
        24  77  77  2Ø              X3      Ø6   2
        25  21  22  23              X4      Ø7   2
        26  24  25  26              A1      1Ø   2
        27  27  3Ø  31              A2      11   2
        31  77  77  55              A3      12   2
        32  56  57  6Ø              A4      13   2
        34  77  77  32              B1      14   2
        35  77  77  33              B2      15   2
        36  77  77  34              B3      16   2
        37  77  77  35              B4      17   2
        38  77  77  36              FOT      Ø   3
        39  77  77  37              AOT      1   3
        4Ø  77  77  4Ø              BOT      2   3
        41  77  77  41              SHL      3   3
        42  77  77  42              SHR      4   3
        43  77  77  43              SQL      5   3
        44  77  77  44              SQR      6   3
        45  77  77  45              LOG      7   3
        46  77  77  46              FORCE   ØØ   5
        47  77  77  47              AANDB   Ø1   5
        48  77  77  5Ø              DANDB   Ø2   5
        49  77  77  51              AORB    Ø3   5
        5Ø  77  77  52              DORB    Ø4   5
        51  77  77  53              AXORB   Ø5   5
        52  77  77  54              DXORB   Ø6   5
        53  77  77  61              INVRA   Ø7   5
       1Ø9  77  4Ø  41              INVRD   1Ø   5
       11Ø  42  43  44              INVRB   11   5
       111  45  46  47              INVRQ   12   5
       119  77  5Ø  51              NOTA    13   5
       12Ø  52  53  54              NOTD    14   5
                                    NOTB    15   5
```

SAMPLE
MICROCODE
ASSEMBLY
LISTING

```
*                                                            *
*     ENVIRONMENTAL TEST PROGRAM MACROS                      *
*                                                            *
**************************************************************
$CMH
STTUP FORCE      PC PC STTA        1110000000011000001      1517 00000073360030100
$CML
STTA  FORCE      PC PC P           1110000000011000001      0355 00000073560030100
      TRAD       B4 B4 P           0010100000001100001      0356 40177473624014100
      TRAD       PC PC P           0010100000001000001      0357 40000074024010100
      TRAA       B4 B4 P           0010010000011100001      0360 36177474222034100
      TRAD       X2 X2 P           0010100000001000001      0361 40052474424010100
      TRAA       B4 B4 P           0010010000011100001      0362 36177474622034100
      TRAD       A3 A3 FETCH       0010100000001000001      0363 40125000424010100
$CMH
TFET  DECRB      PC PC TFEA        0010000000001000001      1520 52000075020010100
$CML
TFEA  FORCE SHR  R2 R2 P           0010000000011000001      0364 01021075220030100
      INVRA      R2 R1 P           0010000000011000001      0365 16020475420030100
      NOPB       B4 B4 P           0010000000001000001      0366 42177475620010100
      TRAA  AOT  X2 X2             1100011000010100001      0367 36252400143024100
$CMH
PROM  DECRD      B4 B4 PROMA       1010100000011100001      1521 50177476124034100
$CML
PROMA TRAD       X3 X3 P           0010100000001000001      0370 40063076224010100
      FORCE      X4 X4 P           0010000000011000001      0371 00073476420030100
      TRAA       X4 X4 P           0010000000011000001      0372 36073476620030100
      TRAA       B4 B4 P           0010010000011100001      0373 36177477022034100
      TRAD       A1 A1 P           0010100000001100001      0374 40104077224014100
      FORCE      A2 A2 P           0010000000011000001      0375 00114477420030100
      FORCE      R3 R3 P           0010000000011000001      0376 00031477620030100
      FORCE      X1 X1 P           0010000000011000001      0377 00042100020030100
PRA   TRAD  SHR  B4 B4 P           0010100000001000001      0400 41177500224010100
      TRAA  SHR  R3 B3 P           0010000000001000001      0401 37037100420010100
      ADDAB      R3 B4 P           0010000000001000001      0402 56167500620010100
      TRAA  SHL  B4 B3 PRC         1010000000101000001      0403 36777101120050100
$CMLE
PRC   TRAA       X1 X1 PRD         1010000000011000001      0404 36042102720030100
      TRAA  SHL  B4 B3 P           1010000000111000001      0405 36777101520070100
$CMLE
      TRAA       X4 A2 PRJ         1010000000001000001      0406 36074502120010100
      NOPB       B4 B4 PRD         1010000000001000001      0407 42177502720010100
PRJ   DANDB      A2 A2 P           1010000000001000001      0410 04114502320010100
      AANDB      R3 A2 PRG         0010000010101000001      0411 02034502420250100
$CMLE
PRG   NOPB       B4 B4 PRC         1010000000001000001      0412 42177501120010100
PRD   ADDDB      R3 R3 P           1010100000001000001      0413 60031503124012100
      DECRB      X3 X3 PRH         0010000010101000001      0414 52063103420250100
$CMLE
```

FIRMWARE

First DAP application was microcoded-
No macro set defined

Microcode adds flexibility/ extra processing power

BUT

Application was hand coded!

Microcoding is not a reasonable task!

NEEDED

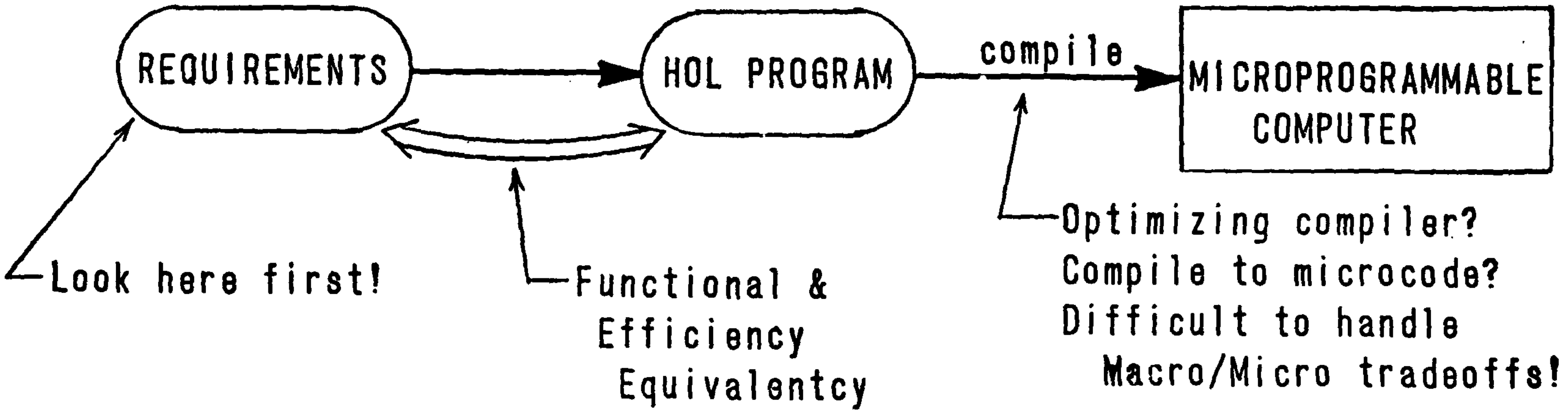
A "good" Macro Instruction Set

MACRO / MICRO DILEMMA

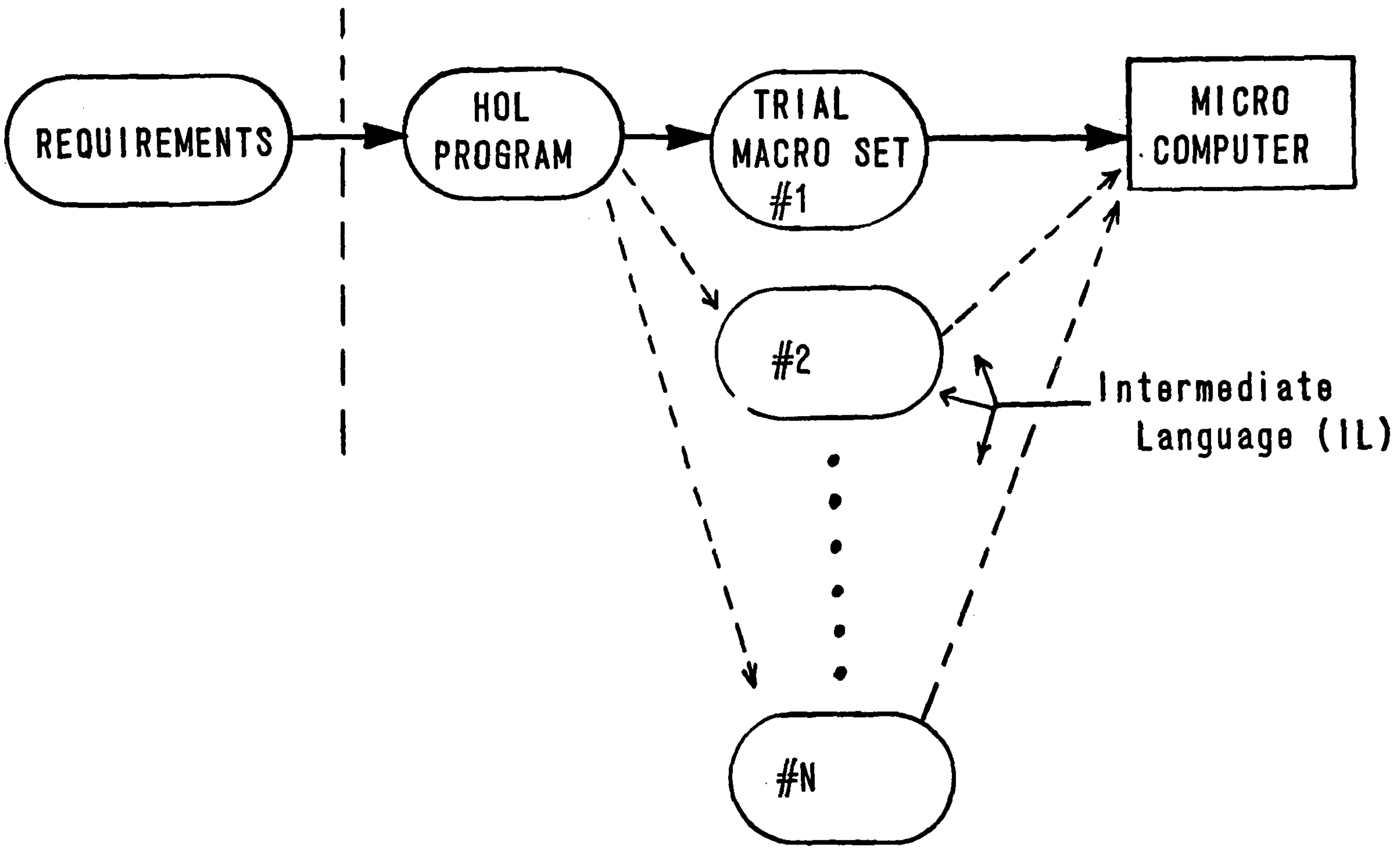
PROBLEM: Find a "*good*" Macro Instruction Set for the application.

What partitioning between main and microprogram memory?
How to develop software and firmware to produce
a highly optimal hardware utilization?

SOLUTION:


REQUIREMENTS
HOL PROGRAM
compile
MICROPROGRAMMABLE COMPUTER
Look here first!
Functional & Efficiency Equivalentcy
Optimizing compiler?
Compile to microcode?
Difficult to handle Macro/Micro tradeoffs!

COMPILATION



SAMPLE HOL LISTINGS

Structured Source

```
  DECLARE Ø: INTERNAL
  DECLARE BPHI: VARIABLE
  DECLARE FLAG1: VARIABLE
  DECLARE ROL: VARIABLE
  DECLARE X11: VARIABLE
  DECLARE X12: VARIABLE
  DECLARE SWCH1: VARIABLE
  DECLARE BPHIC: VARIABLE
  DECLARE EX1P: VAPIABLE
  DECLARE AØ8: CONSTANT 1.3621
  DECLARE AØ9: CONSTANT Ø.1168
  DECLARE BØ8: CONSTANT Ø.5
  DECLARE BØ9: CONSTANT Ø.98763
C
C DO-NOTHING PROGRAM SEGMENT
C     (DATA DECLARATIONS ARE      SHOWN)
C
  IF(FLAG1.EQ.Ø)
     THEN
        X12=AØ8*BPHI-BØ8 *ROL - X11
        X11=AØ9*BPHI+BØ9*ROL
     ELSE
        IF(SWCH1.LT.Ø)
           THEN
              BPHI=BPHI+BPHIC
        ENDIF
        FLAG1=Ø
  ENDIF
  CALL EXEC1(EX1P)
  END
```

Compiler Listing

```
 3          DECLARE Ø: INTERNAL
 4          DECLARE BPHI: VARIABLE
 5          DECLARE FLAG1: VARIABLE
 6          DECLARE ROL: VARIABLE
 7          DECLARE X11: VARIABLE
 8          DECLARE X12: VARIABLE
 9          DECLARE SWCH1: VARIABLE
10          DECLARE BPHIC: VARIABLE
11          DECLARE EX1P: VARIABLE
12          DECLARE AØ8: CONSTANT 1.3621
13          DECLARE AØ9: CONSTANT Ø.1168
14          DECLARE BØ8: CONSTANT Ø.5
15          DECLARE BØ9: CONSTANT Ø.98763
16  C
17  C       DO-NOTHING PROGRAM SEGMENT
18  C           (DATA DECLARATIONS ARE      SHOWN)
19  C
2Ø  C.      IF(FLAG1.EQ.Ø)
21          IF(FLAG1)9999,8999,9999
22  8999    CONTINUE
23  C.      THEN
24          X12=AØ8*BPHI-BØ8 *ROL - X11
25          X11=AØ9*BPHI+BØ9*ROL
26  C.      ELSE
27          GO TO  9998
28   9999   CONTINUE
29  C.      IF(SWCH1.LT.Ø)
3Ø          IF(SWCH1)8997,9997,9997
31  8997    CONTINUE
32  C.      THEN
33          BPHI=BPHI+BPHIC
34  C.      ENDIF
35   9997   CONTINUE
36          FLAG1=Ø
37  C.      ENDIF
38   9998   CONTINUE
39          CALL EXEC1(EX1P)
4Ø          END
```

GRAPHICS

Dr. N. Radhakrishnan
WES, COE

GRAPHICS

SESSION CHAIRPERSON: Dr. N. Radhakrishnan

U.S. Army Engineer Waterways Experiment Station

SESSION SUMMARY

This session addressed graphics standardization and the application of graphics to color and 3-D geometry. The topics presented showed that graphics is an important design tool. It is a tool that saves both time and money by presenting data in the engineer's language-graphics. Since graphics can and does play an important role in engineering design, there is a concern throughout the graphics user community of using various software packages. A standard graphics package will provide both program and prorammer portability which can translate into dollar savings in both training and dissemination of information. The four topics that have been presented were:

GRAPHICS STANDARDS by Bertram Herzog, University of Colorado

GCS AND GRAPHICS STANDARDIZATION IN THE ARMY CORPS OF ENGINEERS
by James M. Jones, II, U.S. Army Corps of Engineers

3-D GEOMETRY GENERATION WITH A PRACTICAL APPLICATION
by Fred T. Tracy, U.S. Army Corps of Engineers

COHERENCE CONCEPTS IN COMPUTER SYNTHESIZED REAL-TIME DISPLAYS
by John Staudhammer, North Carolina State University

Graphics Standards

Bertram Herzog
Director, University Computing Center

University of Colorado

The field of computer graphics is reaching a degree of maturity that workers in the field are seeking to draft a proposed standard for computer graphics--especially software. Several national and international standards groups are working on this problem. In the United States, such activities are sponsored by ANSI and the Graphics Standards Planning Committee, GSPC, of the Association of Computing Machinery's Special Interest Groups for Graphics, SIGGRAPH. This latter group, GSPC, has drafted a proposed standard, often called Core. Several authors have reported on their implementation of this draft. Revision of the Core, in response to *user critiques, is under way.*

A description of the essential elements of the Core was presented. Examples of programs written using one of the implementations illustrated the features thought to be ready for standardization.

Finally, a report of current standards activities was given with an appeal for user participation to ensure that a good standard will evolve.

NOTES ON GRAPHICS STANDARDS

The talk is primarily a progress report of the Graphics Standards Planning Committee of ACM's SIGGRAPH. The reader is urged to consult COMPUTER GRAPHICS, A Quarterly Report of SIGGRAPH-ACM, Vol. 11, Numbers 3-4, Fall 1977 for a report of the major accomplishments which include a review of several graphics packages and the specification of the CORE proposal.

This CORE proposal has resulted in several trial implementations. The implementors reported their work at a session of SIGGRAPH 78 in Atlanta in August 1978. Their papers are published in COMPUTER GRAPHICS, Vol. 12, Number 3, August 1978. The important titles are:

TIGS An Overview of the Terminal Independent Graphics System

by Robert L. Heilman, Batelle Columbus Labs, Columbus, Ohio and
Jean M. Marchant, Control Data Corporation, Arden Hills, Minnesota

Core Standard Graphic Package for the VGI 3400

by Ken Levine, Vector General, Inc., Woodland Hills, California

DIGRAF - A FORTRAN Implementation of the Proposed GSPC Standard

by James R. Warner, Margaret A. Polisher, and Robert N. Kopolow,
University of Colorado Computing Center, Boulder, Colorado

An Implementation of the ACM/SIGGRAPH Proposed Graphics Standard in a Multisystem Environment

by Richard G. Kellner, Theodore N. Reed, and Ann V. Solem, Los Alamos
Scientific Laboratory, Los Alamos, New Mexico

A Microprocessor-Assisted Graphics System

by Griffith Hamlin, Jr., Los Alamos Scientific Lab, Los Alamos, N.M., and
Thomas Crockett, NASA Langley Res Center, Hampton, Virginia

A Flexible, High Performance Interactive Graphics System

by Roger J. Hubbold and P.J. Bramhall, University of Manchester,
Manchester, England

A more complete review of the previously reviewed graphics packages together with a comparison to the CORE is given in a paper edited by Ewald and Fryer entitled, Final Report of the GSPC State-of-the-Art Subcommittee and is published in COMPUTER GRAPHICS, Volume 12, Numbers 1-2, June 1978. In this publication can be found reports about the ANSI activities and the plans for GSPC for 1978-79. The following pages are reproduced with permission of ACM.

ACM/SIGGRAPH GSPC

Goals and Activities*

Introduction

The second Winter Meeting of the Graphics Standards Planning Committee, GSPC, was held at the University Computing Center, University of Colorado, Boulder, Colorado in March 1978. Twenty people, in the role of GSPC members, implementors, and observers attended the meeting.

The meeting had two main purposes. The first was to hear reports on relevant graphics activities that had taken place following the publication of the GSPC Core System design. In particular, constructive comments were made by current implementors of the Core and similar graphics systems. The second purpose of the meeting was to define future goals and activities for the GSPC, and to reorganize the membership into appropriate working groups. These definitions and organizational plans were produced by a workshop approach--many subgroup meetings were held to develop recommendations that were then considered during reconvenings of the whole group.

Meeting topics and results believed to be of general interest to SIGGRAPH members are summarized in the remaining sections of this report.

Current Goals and Organization

The major goals of the GSPC are:

1. To continue technical studies appropriate to computer graphics standardization.

2. To disseminate information resulting from the studies, interface with other standardization groups, and establish dialogs within the general graphics community.

The current co-chairmen of the GSPC are:

Bob Heilman
Battelle Columbus Labs
505 King Avenue
Columbus, OH 43201
614/424-7340

Bert Herzog
Computer Center
University of Colorado
Boulder, CO 80309
303/492-4331

Several "standing" subcommittees of the GSPC are:

1. Speakers' Bureau:

   Bert Herzog, Chairman

   This bureau is a mechanism for providing speakers to organizations seeking information about GSPC activities.

2. Education and Publicity:

   Tim Dreisbach, Chairman
   SofTech, Inc.
   460 Totten Pond Road
   Waltham, MA 02154
   617/890-6900

*This summary is primarily composed of extracts taken from the Report of the GSPC Winter Meeting, 1978. That report was submitted by GSPC Co-Chairmen Bob Heilman and Bert Herzog and was based on a draft version written by GSPC members Peter Bono, Tim Dreisbach, and Jim Michener.

This is a relatively new activity in response to the GSPC's important role in providing continuing education to the graphics community on the importance of standards and current standardization activities. Some items in need of publicity, to all types of audiences, are graphics standards efforts in general, existing GSPC outputs, and current GSPC activities.

3. Core System Comments and Review:

   Margaret Polisher, Chairman
   Computing Center
   University of Colorado
   Boulder, CO 80309
   303/492-6501

   This activity provides a formal mechanism for handling questions and comments on the GSPC Core system. Correspondence received by Margaret will be forwarded to the appropriate GSPC members or subgroups. As in the past, replies will be coordinated and sent to the initiating correspondent. Margaret is beginning to work with Tim Dreisbach on an effort to publish frequently occurring comments or ones with important technical implications, together with their corresponding responses. If there is enough interest, this could be accomplished through brief, but regular, sections in Computer Graphics.

4. Implementation Surveys:

   Peter Bono
   Naval Underwater System Center
   Code 314
   New London, CT
   203/442-0771 (x2754)

   This activity provides a clearing house for information about current implementations of the Core System and similar software packages. Implementation efforts have, and are, being surveyed and cataloged.

5. ANSI Coordination:

   Bob Heilman, Chairman

   This activity provides a formal mechanism for coordinating the technical efforts of the GSPC with the activities of the ANSI graphics study group. A rich dialog is expected because there are numerous individuals who are members of both groups.

   As a result of the Winter Meeting, a number of additional subcommittees were organized to pursue technical topics.

6. Core System Refinements:

   Vic Wallace, Co-Chairman
   Computer Science Dept.
   University of Kansas
   Lawrence, KA 66044
   913/864-4482

   Elaine Sonderegger, Co-Chairman
   2615 Sixth Street #J
   Santa Monica, CA 90405
   213/822-1511

   The purpose of this group will be to reconsider portions of the Core System design in response to comments and criticisms. It will expand on portions of the design that were not fully defined in the 1977 Status Report. Particular issues already identified are text and other output primitives, input requirements, segmentation and control, output device-coordinate systems, etc. The group's long-term goal is the publication of a revised Core System design document by August 1979.

7. Core System Extensions:

Lansing "Chip" Hatfield, Chairman
Lawrence Livermore Labs
P.O. Box 808, MS L-156
Livermore, CA 94550
415/411-8567

The purpose of this group is to ensure that the Core System will not preclude effective use of computer graphics capabilities as they now exist or will likely develop in the near future. Some appropriate subjects to be investigated are raster graphics, color, conic functions, high-performance devices, and distributed processing systems. Working documents for at least the color/raster areas are being drafted for review by SIGGRAPH '78.

8. Core System Partitioning and Protocols:

Andy Goodrich
Computing Center, North Campus
University of Michigan
Ann Arbor, MI 48109
313/764-2121

This group will be investigating methodologies for partitioning the Core System. As one example, criteria based on program structure and portability issues could be used to identify a basic kernel of routines within the currently rich system.

Other topics to be considered are actual specifications of kernel routines, specification of other system partitionings, possible pseudo-display code formats and protocols, etc.

9. Graphics Program Structure, Techniques, and Higher-Level Requirements:

Tim Dreisbach, Acting Chairman

The goal of this activity is to relate software engineering methodologies to (interactive) computer graphics and to determine if specific programming guidelines can be defined. A possible topic to be considered is how the notions of well-structured programs are specifically reflected in graphics software; for example, can "Elements of Graphics Programming Style" be identified? Other areas that can be addressed are how graphics programs can be designed to improve user interfaces, i.e., implementors of common graphics functions (linear algebra for viewing transformations), and higher-level requirements (e.g., hierarchical modeling systems, graph and plot generation systems).

A focused direction for this group cannot be determined until its membership and leader are identified. Tim Dreisbach is serving as acting chairman until a permanent individual is appointed; volunteers are encouraged to step forward.

10. Foundation of Computer Graphics Standards:

Jon Meads, Chairman
Tektronix, Inc.
P.O. Box 500
Beaverton, OR 97077
503/682-3411 (x2229)

This group is to investigate the role of graphics standards over the long term. Planned areas of work include a forecast of technological advances and an analysis of their impact on graphics systems, hypothesizing what people really need and how computer graphics can help, prophesying new areas where graphics can be effectively utilized, and developing a taxonomy of man-machine interactions from functional and applications viewpoints. The committee will review the other GSPC activities based on these considerations.

Because the five technical subcommittees are newly formed, their memberships are currently open. All interested individuals are encouraged to contact either the GSPC co-chairmen or specific subcommittee chairmen.

GSPC/ANSI Relationship

As mentioned previously, a close relationship exists between these two groups, with several individuals belonging to both. During the ANSI Study Group meeting in February, Peter Bono served as an official liaison representative of the GSPC. The ANSI members were invited, and many participated, as observers to the Winter Meeting of the GSPC. The GSPC regards itself as a technical and educational body that can provide a knowledge base in support of the ANSI group's activities.

A concern was noted among some ANSI group members regarding the momentum of the GSPC Core System and the potential for a resulting defacto standard. There is general consensus among the GSPC people that premature "freezing" of the current Core System proposal is highly undesirable. The GSPC has no "official" authority, nor any desire, to legislate standards. However, both groups recognize the possibility that standards development recommendations by the ANSI group might include a suggestion that the ACM, SIGGRAPH, or even GSPC specifically, be authorized as a technical development body.

Existing and Forthcoming Papers

The 1977 Status Report of the GSPC appeared in Volume 11, Number 3 (Fall 1977) of Computer Graphics. It contained a summary of the State-of-the-Art Survey and the functional specification of a proposed Core System. Additional copies can be ordered, prepaid, from:

Association for Computing Machinery, Inc.
P.O. Box 12105
Church Street Station
New York, NY 10249

ACM and SIGGRAPH members: $ 9.00
All others: $12.00

ACM SIGGRAPH has endorsed magnetic tapes of the Status Report that can be obtained from:

ATTN: SIGGRAPH GSPC 1977 Status Report
University Computing Center
University of Colorado
Boulder, CO 80309

PRICE: $50.00/tape

An informal GSPC presentation is currently being planned as one part of the SIGGRAPH '78 poster session. The goals are to discuss technical issues related to the GSPC report in small-group fashion, to inform interested individuals about the current GSPC status, and to solicit comments and volunteers. Topics related to the activities of specific technical subcommittees (perhaps as reflected in draft working papers) can be addressed.

A compatible set of papers on topics related to the Core System design has recently been produced. These papers were not directly done as a GSPC activity, but were written by several members of the original Core System design group. The papers are scheduled to appear in the December issue of ACM Computing Surveys (Vol. 10, No. 4).

On-going Implementations

Last December and January, a questionnaire was sent out to 22 people known to be working on, or considering, a design or implementation of the Core System. Fourteen replies were received.

The envisioned Host Environments included equipment from eight to ten computer manufacturers, seven to eight programming languages, and approximately twenty different display devices. Equal interest was expressed in all four levels of the proposed Core System.

Machine- and operating system-independence of the implementation was unanimously deemed "very important." Language independence was much less important, with nine responses in the "not very important" category and only four judging it "important." Importance of device driver portability was split. Seven felt that it was important, five not important, and two expressed no opinion.

Four principal areas of concern emerged from the Remarks Section of the questionnaire:

1. Device-Independent Pseudo-Display File Representation

2. Device-Independent/Device-Dependent Interface

3. Device Driver Portability

4. Storage Allocation and Memory Management Problems

Seven implementors and one implementation designer accepted invitations to describe their efforts at the GSPC's Winter Meeting. Richard Fryer of the China Lake Naval Weapons Center described a Level 2, FORTRAN, mini-computer system. Ted Reed of Los Alamos Scientific Laboratory described the Common Graphics System--a joint approach to graphics for Sandia, LASL, and the Air Force Weapons Laboratory. Jim Warner of the University of Colorado Computing Center described DIGRAF--"Device-Independent Graphics for FORTRAN." Peter Sih of the IBM Los Angeles Scientific Center described a non-product-oriented implementation of the Core System accessible from APL. Dick Puk of Sandia Laboratories described the BGP-- Basic Graphics Package--Level 2 implementation of the Core System and its relationship to the Common Graphics System (CGS) described by Ted Reed. Dave Verhoeven of Tektronix Inc., indicated that industry has some problems contributing to standardization because of the proprietary nature of their work. Although the 1977 Core System definition serves as a focus for internal development, there is a hesitation to create a product too soon, in light of possible differences in the eventual standard. Andy Goodrich of the University of Michigan described an implementation of the Core System built on top of an existing system, IG, having a hierarchical picture structure. Jim Michener, of Intermetrics, Inc., proposed changes to the Core System derived during the design for the interface between device-independent and portions of an implementation.

GCS & Graphics Standardization in the Army Corps of Engineers

James M. Jones, II
U.S. Army Engineer Waterways Experiment Station

This paper described the Army Corps of Engineers (COE) involvement in computer graphics and the role of the Graphics Compatibility System (GCS) in the development of a standard graphics software system for COE now and in the future.

The Corps has long been involved in passive graphics -- producing a plot either on-line or off-line. This often meant plotting a certain data set several times to eliminate all input errors. It was only with the development of interactive graphics that the Corps' engineers could interact with a plot and correct any errors prior to producing a final plot.

When the Corps became initially involved in interactive graphics, it was recognized that a graphics software system was needed that could support several devices (passive and interactive). This would provide both computer program and programmer portability. The Graphics Caompatibility System (GCS) developed and supported by West Point Military Academy/Computer System Command/Army Material Command was selected.

The Waterways Experiment Station (WES) Automatic Data Processing (ADP) Center developed several applications using GCS. As the use of GCS expanded to several Corps' offices, graphics training courses were provided for Corps' personnel. When West Point was no longer able to support GCS, administrative support and maintenance of GCS was transferred to the WES ADP Center.

GCS was enhanced and modified to support new state-of-the-art graphics techniques. Both a two-dimensional and a three-dimensional version of GCS are available and can be used with several graphics devices on different computers. New capabilities include data structures, segmentation, pseudo display files, and Hershey software character fonts.

As the result of a Corps-wide Graphics Colloquium, GCS will soon become the official graphics software standard for the Corps. With Corps' use and input from a newly formed Users' Group, GCS will continue to evolve and provide a graphics standard that will not confine or restrict the Corps' graphics users. Future work includes support for interactive color graphics new input features, and hidden line/surface capability.

GSC and GRAPHICS STANDARDIZATION in the ARMY CORPS OF ENGINEERS

James. M. Jones

Introduction

The Army Corps of Engineers has been actively involved in passive computer graphics since the early 1960's. It was widely recognized that passive graphics:

a. Eliminated massive paper output

b. Gave the engineer results in a form he could readily understand (graphs)

c. Saved time and money

Although passive graphics provided the engineer plots of his data, it was still a tedious process because he could not interact with his data as it was plotted. The engineer would view the plot, detect some data errors, correct those errors, and re-submit the corrected data to produce another plot. This process was repeated until an error-free plot was produced. An example of this process is the use of the finite element method. The finite element method assumes that the forces that interact in a complex structure can be calculated by subdividing the structure into a series of small, finite pieces, and computing the forces in each piece. For a finite element analysis an engineer used to:

a. Manually divide the structure into small, finite elements.

b. Provide element numbers, node numbers and x, y, z coordinates for the nodes.

c. Punch this input data.

d. Pre-process the input data and produce a plot.

e. Repeat steps b-d to eliminate any data errors.

f. Apply the error-free input data to an analysis program.

g. Post-process the output data and produce a plot.

h. Repeat steps a-g until "good" output results.

Interactive Graphics and GCS

In 1972, the Waterways Experiment Station (WES) ADP Center purchased a Tektronix 4012 storage tube terminal to provide interactive computer graphics. It was felt that interactive graphics would result in:

a. Greater productivity of engineers' time.

b. Better design of structures

c. Better analysis of complex structures

d. More engineers using computers in their work

Now the engineer could interact with his plot, correct any errors and produce one final, error-free plot.

There were several graphics software packages available that supported interactive computer graphics. To prevent duplication of resources (people and money), a survey was conducted to determine if one graphics software package could support both passive and interactive graphics on several graphics devices. Corps offices that were involved in graphics were visited to determine existing and potential graphics requirements. Visits were made to other federal agencies, commercial vendors, universities and colleges, and several experts in computer graphics. The Graphics Compatibility System (GCS), developed and supported by the West Point Military Academy, Computer Systems Command and Army Material Command was selected. The reasons were:

a. Supported state-of-the-art graphics concepts

b. All subroutines written in FORTRAN

c. Operational on Honeywell computers

d. Supported multiple graphics devices

e. Non-propriety software package

f. West Point/Computer Systems Command maintenance and support of the system

g. It had been field tested and used by other agencies

After selecting GCS, WES obtained funding from the Office of the Chief of Engineers (OCE) to develop civil engineering applications. WES obtained "seed money" from Project Idea (Army Material Command) to develop pre- and post-processors for the finite element method.

Mr. Fred Tracy, WES, developed a 2-D pre- and post-processor with GCS and a Tektronix 4012/4014 graphics terminal. The engineer uses the crosshairs on the terminal to interact with his input boundary data. The program automatically generates a finite element grid and numbers the nodes and elements. The final grid can be developed and displayed on the Tektronix terminal. A cost analysis comparison (1975) with the earlier passive graphics technique shows the advantages of interactive graphics.

	Passive	Interactive
Labor	$ 950	$ 260
Computer	120	320
Plotter	6	6
	$1076	$ 586

This is representative of the savings achieved with interactive graphics.

GCS was used in developing other graphics applications. As the use of GCS expanded to several Corps' offices, graphics training courses were provided for Corps' personnel. In 1975, West Point was directed to no longer support GCS and in January 1976, administrative responsibility for GCS support and maintenance was transferred from West Point to the WES ADP Center.

GCS Status

While graphics applications were being developed, the initial 2-D version of GCS was enhanced and modified to support 3-D graphics with new state-of-the-art graphics techniques. Among these were data structures, segmentation, psuedo display files, color, and Hershey character fonts. Figures 1-6 illustrate several GCS features.

GCS was favorably evaluated by the "GRAPHICS STANDARDS PLANNING COMMITTEE, STATE OF THE ART SUBCOMMITTEE, ACM" in a graphics system comparison document. Their general comments are: [1,2,3]

> The Graphics Compatibility System (GCS) package is a set of ANSI FORTRAN-IV - callable subroutines unified through a named common area called Graphics Status Area. The system attributes and control options are provided through this area. The software provides a display of two-dimensional data and the extensions allow three-dimensional data.
>
> The software interface is provided in several basic levels:
>
> 1. Simple graphics input/output
> 2. Window-viewport management with secondary axis control
> 3. Graphing and analysis output
> 4. Buffer management for dynamic refresh systems
>
> The calling sequences are short and consistent. The basic modes of operation are controlled through two routines (USET, UPSET) that set the modes and values for the central management interface, which is the shared common area.

New documentation has been published. To support training of Corps' personnel in GCS, several computer-aided-instruction (CAI) lessons have been developed. There are two versions of the lessons - one provides text output at an alphanumeric terminal (Figure 7) and the other provides both text and graphics output when used on a

Tektronix 4014 graphics terminal with hardcopy unit (Figure 8).

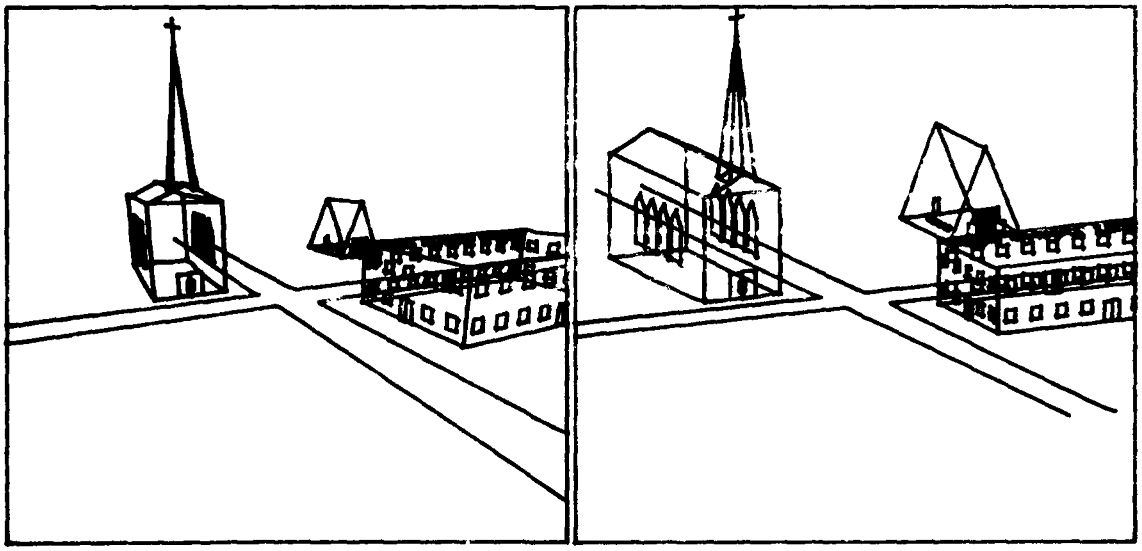
Standardization

On 1-3 August 1978, a Corps-wide graphics colloquium was held to exchange information between the different Corps offices. Ninety-four Corps' individuals attended and it was determined that there needs to be a standard graphics software package for Corps-wide use. GCS will be the basis for this standard and an EN (engineering regulation) is in the draft stage. Not only has use of GCS grown in the Corps, but there are over 150 government agencies, colleges and universities, and private industries that have obtained GCS. A GCS Users Group was formed at Siggraph (Special Interest Group on Graphics) in Atlanta, Georgia, 22 August 1978 to support their needs. This organization will be a significant driving force in standardization efforts.

Future

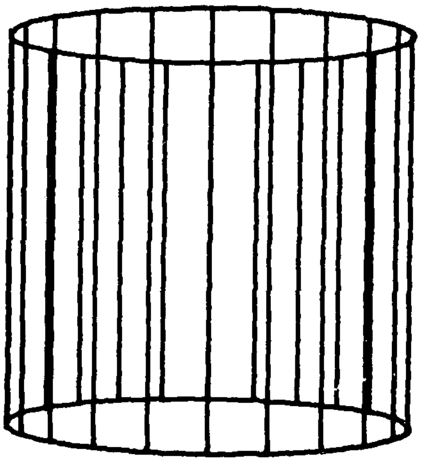
With continued support and use by the Corps and input from the newly formed Users Group, GCS will continue to evolve and provide a graphics standard that will not confine or restrict the Corps' graphics users. Future work includes support for raster graphics[4], new input features and hidden line/surface capability.

```
CALL ATTACH (6,'/SAVE;',3,0,IST,)
CALL USTART
CALL UPSET ('LIBRARY FILE',1.)
CALL UTILTY ('LOAD LIBRARY FILE',6.)
CALL UPSET ('TERMINATOR','<')
CALL USET ('VIEW DISTANCE')
CALL UVWPLN (150.)
CALL UWINDO (-100.,100.,-100.,100.)
CALL UVIEW (-70.,-150.,50.,0.,0.,10.)
CA! L UDAREA (0.,7.,3.,10.)
CALL UOUTLN
CALL USET ('PERSPECTIVE')
CALL UINVOK ('VILLAG')
CALL UDAREA (7.1,14.1,3.,10.)
CALL UOUTLN
CALL UVIEW (-70.,-150.,50.,0.,0.,10.)
CALL USET ('ORTHOGRAPHIC')
CALL UINVOK ('VILLAG')
CALL USET ('EXTRALARGE CHARACTERS')
CALL USET ('DEVICE UNITS')
CALL USET ('ACENTER')
CALL UPRINT (7.,.7,'PROJECTION MODES<')
CALL UPRINT (7.,.3,'FIGURE 1<')
CALL UEND
STOP
END
```



PROJECTION MODES
FIGURE 1

```
    CALL USTART
    CALL UDAREA (2.,12.,.9,1Ø.Ø)
    CALL UVIEW (15Ø.,15Ø.,3Ø.,Ø.,Ø.,3Ø.)
    CALL UWINDO (-8Ø.,8Ø.,-8Ø.,8Ø.)
    CALL USET ('CYLINDRICAL')
    CALL UCRCLE (Ø.,Ø.,3Ø.)
    CALL UPSET ('ZVALUE',6Ø.)
    CALL UCRCLE (Ø.,Ø.,3Ø.)
    DO 1Ø I=1,346,15
    THETA = I - 1
    CALL USMOVE (3Ø.,THETA,Ø.)
 1Ø CALL USPEN (3Ø.,THETA,6Ø.)
    CALL USET ('EXTRALARGE')
    CALL USET ('DEVICE')
    CALL USET ('ACENTER')
    CALL USET ('RECTANGULAR')
    CALL UPRINT (7.,.7,'CYLINDRICAL COORDINATES\')
    CALL UPRINT (7.,.3,'FIGURE 2\')
    CALL UEND
    STOP
    END
```

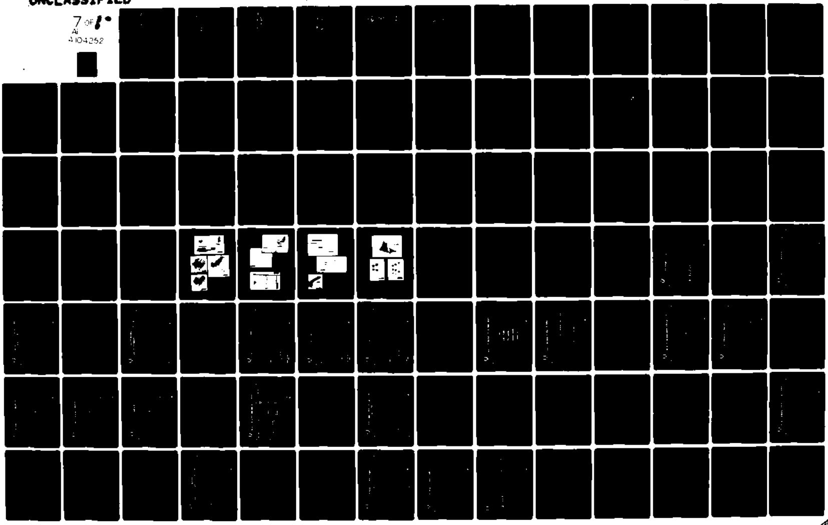


CYLINDRICAL COORDINATES
FIGURE 2

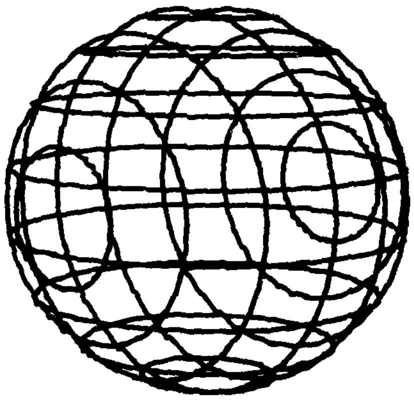
AD-A104 252 INTERNATIONAL BUSINESS SERVICES INC WASHINGTON DC F/G 9/2
US ARMY SOFTWARE SYMPOSIUM (2ND) HELD AT WILLIAMSBURG, VIRGINIA--ETC(U)
1978 S M TAYLOR DAAK70-78-D-0030

UNCLASSIFIED NL

7 OF [illegible]
AD
A104252

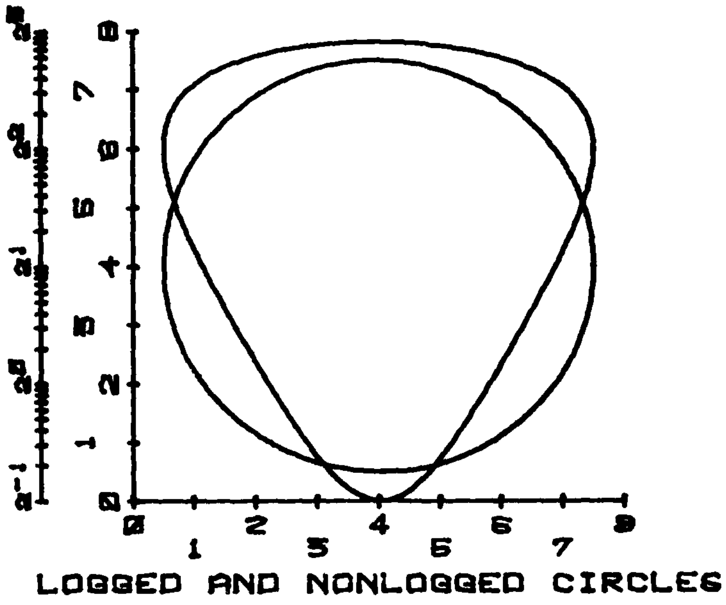


```
      CALL USTART
      CALL UDAREA (2.,12.,.0,10.0)
      CALL UVIEW (150.,150.,30.,0.,0.,0.)
      CALL UWINDO (-60.,60.,-60.,60.)
      CALL USET ('SPHERICAL COORDINATES')
      RADIUS = 30.
      ILOOP = 0
   10 IF (ILOOP .EQ. 2) GO TO 20
      DO 15 I = 1, 181, 20
      RHO = I - 1
      CALL USMOVE (RADIUS,0.,RHO)
      DO 15 J = 1, 361, 10
      THETA = J - 1
   15 CALL USPEN (RADIUS,THETA,RHO)
      CALL USMOVE (0.,0.,0.)
      CALL USROTA (0.,90.,0.)
      ILOOP = ILOOP + 1
      GO TO 10
   20 CONTINUE
      CALL USET ('ACENTER')
      CALL USET ('EXTRALARGE')
      CALL USET ('RECTANGULAR')
      CALL USET ('DEVICE')
      CALL UPRINT (7.,.7,'SPHERICAL COORDINATES\')
      CALL UPRINT (7.,.3,'FIGURE 3\')
      CALL UEND
      STOP
      END
```



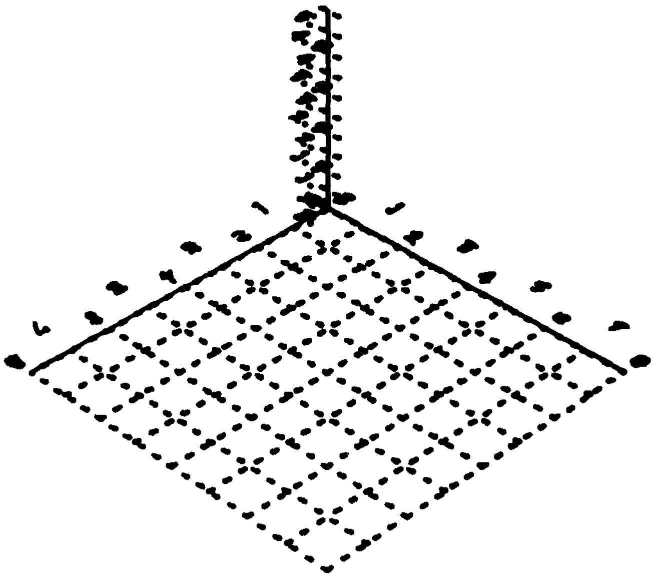
SPHERICAL COORDINATES
FIGURE 3

```
CALL USTART
CALL UDAREA (2.,12.,.0,10.0)
CALL USET ('SOFT')
CALL UPSET ('VERTICAL',.4)
CALL UPSET ('HORIZONTAL',.4)
CALL UWINDO (-3.,10.,-3.,10.)
CALL UPSET ('TICX',1.)
CALL UPSET ('TICY',1.)
CALL UPSET ('XLABEL','LOGGED AND NONLOGGED CIRCLES\')
CALL USET ('XBOTHLABELS')
CALL USAXIS ('XAXIS',0.,0.,0.,8.)
CALL USAXIS ('YAXIS',0.,0.,0.,8.)
CALL UCRCLE (4.,4.,3.5)
CALL USCSYS (0.,2.,0.,1.,2.,1.,0.,0.,0.)
CALL UPSET ('HORIZONTAL',.15)
CALL UPSET ('VERTICAL',.3)
CALL UPSET ('BASE',2.)
CALL USET ('YLOG')
CALL USET ('LOGUSER')
CALL USAXIS ('YAXIS',-1.5,.5,0.,7.5)
CALL UCRCLE (4.,4.,3.5)
CALL USET ('HARDWARE')
CALL USET ('ACENTER')
CALL USET ('EXTRALARGE')
CALL USET ('DEVICE UNITS')
CALL USET ('NOLOG')
CALL UPRINT (7.,.7,'BASIC AXIS CREATION\')
CALL UPRINT (7.,.3,'FIGURE 4\')
CALL UEND
STOP
END
```



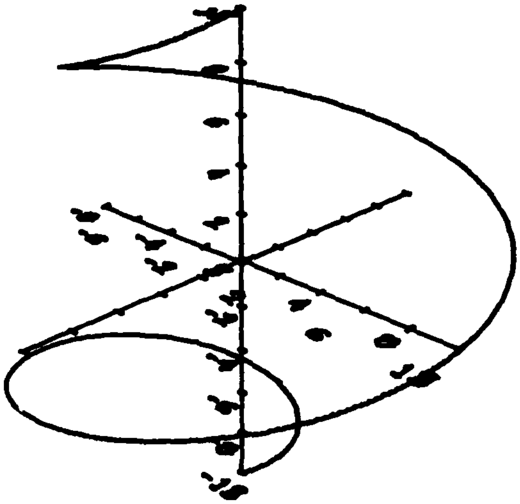
BASIC AXIS CREATION
FIGURE 4

```
CALL USTART
CALL UDAREA (2.,12.,.0,10.0)
CALL UWINDO (-6.,6.,-6.,6.)
CALL UVIEW (30.,30.,30.,2.,2.,0.)
CALL UPSET ('GRID',30.)
CALL USET ('GRID')
CALL USET ('SOFT')
CALL UPSET ('HORIZONTAL',.3)
CALL UPSET ('VERTICAL',.5)
CALL UPSET ('LABEL',90.)
CALL UPSET ('TICP',.3)
CALL UPSET ('TICN',.3)
CALL U3AXIS (0.,6.,0.,6.,0.,5.)
CALL USET ('HARDWARE')
CALL USET ('EXTRALARGE')
CALL USET ('DEVICE')
CALL USET ('ACENTER')
CALL UPRINT (7.,.7,'3-D AXIS CREATION\')
CALL UPRINT (7.,.3,'FIGURE 5\')
CALL UEND
STOP
END
```



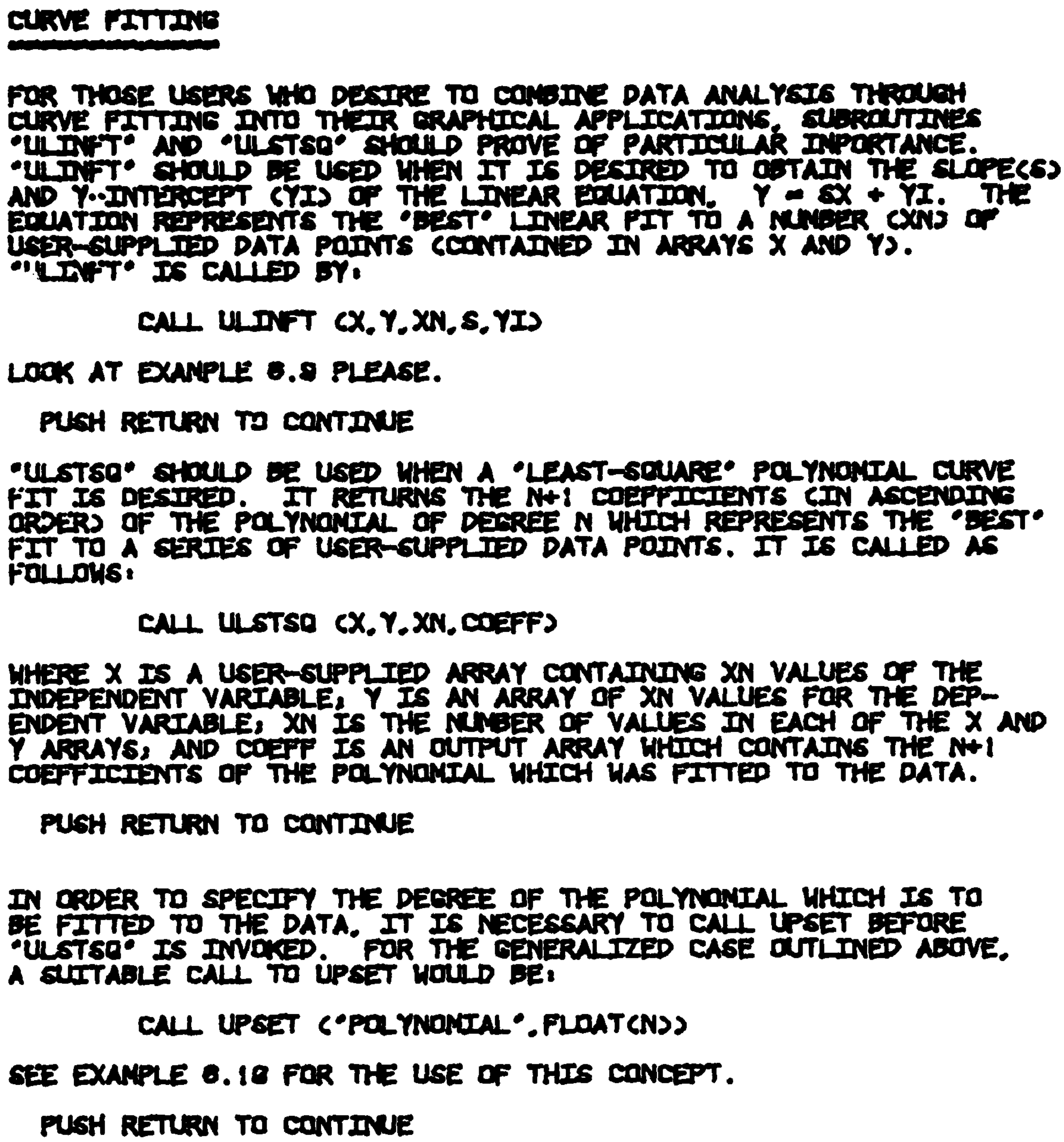
3-D AXIS CREATION
FIGURE 5

```
      REAL X(401), Y(401), Z(401), R
      R = 10.
      DO 10 I = 1, 401
      Z(I) = FLOAT(I-1) * R/200.0 - R
      T = SQRT(R**2 - Z(I)**2)
      X(I) = COS(Z(I) * 0.5) * T
      Y(I) = SIN(Z(I) * 0.5) * T
   10 CONTINUE
      CALL USTART
      CALL UDAREA (2.,12.,.9,10.9)
      CALL UPSET ('HORIZONTAL',.6)
      CALL UPSET ('VERTICAL',1.2)
      CALL USET ('SOFT')
      CALL USET ('NOYLABELS')
      CALL UWINDO (-15.,15.,-15.,15.)
      CALL UVIEW (30.,-30.,20.,0.,0.,0.)
      CALL USPLOT (X,Y,Z,1.,401.,'LNULL')
      CALL UVIEW (0.,0.,150.,0.,0.)
      CALL USET ('ACENTER')
      CALL USMOVE (0.,-11.,0.)
      CALL UPRNT1 ('SPIRAL ON A SPHERE\','TEXT')
      CALL USET ('HARDWARE')
      CALL USET ('EXTRALARGE')
      CALL USET ('DEVICE')
      CALL UPRINT (7.,.7,'3D PLOTTING\')
      CALL UPRINT (7.,.3,'FIGURE 6\')
      CALL UEND
      STOP
      END
```



SPIRAL ON A SPHERE

3D PLOTTING
FIGURE 6

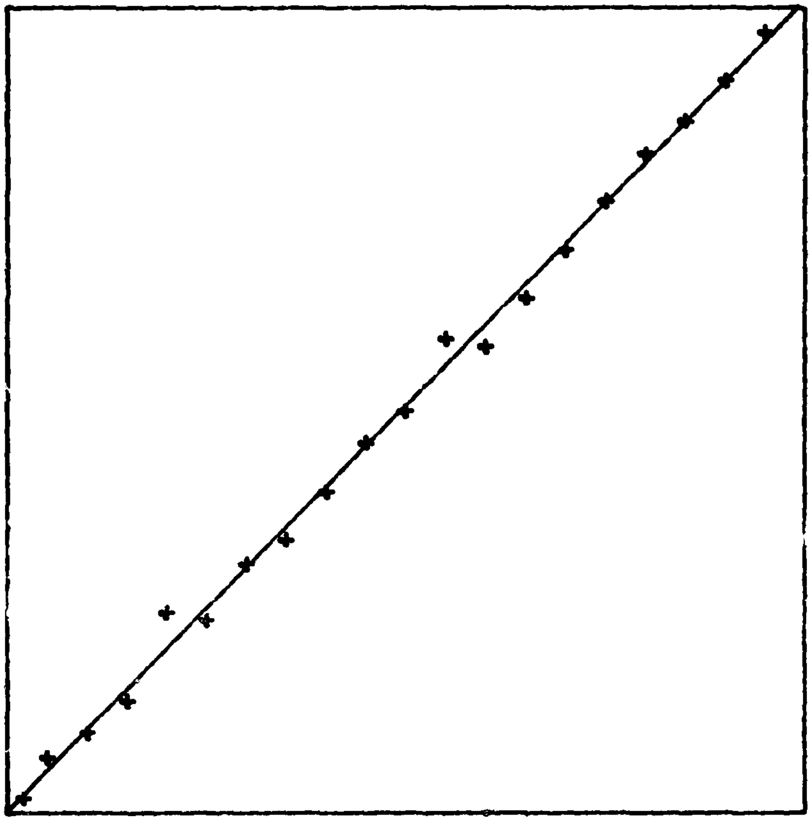


```
CURVE FITTING
-------------

FOR THOSE USERS WHO DESIRE TO COMBINE DATA ANALYSIS THROUGH
CURVE FITTING INTO THEIR GRAPHICAL APPLICATIONS, SUBROUTINES
'ULINFT' AND 'ULSTSQ' SHOULD PROVE OF PARTICULAR IMPORTANCE.
'ULINFT' SHOULD BE USED WHEN IT IS DESIRED TO OBTAIN THE SLOPE(S)
AND Y-INTERCEPT (YI) OF THE LINEAR EQUATION,  Y = SX + YI.  THE
EQUATION REPRESENTS THE 'BEST' LINEAR FIT TO A NUMBER (XN) OF
USER-SUPPLIED DATA POINTS (CONTAINED IN ARRAYS X AND Y).
'ULINFT' IS CALLED BY:

        CALL ULINFT (X,Y,XN,S,YI)

LOOK AT EXAMPLE 8.9 PLEASE.

  PUSH RETURN TO CONTINUE

'ULSTSQ' SHOULD BE USED WHEN A 'LEAST-SQUARE' POLYNOMIAL CURVE
FIT IS DESIRED.  IT RETURNS THE N+1 COEFFICIENTS (IN ASCENDING
ORDER) OF THE POLYNOMIAL OF DEGREE N WHICH REPRESENTS THE 'BEST'
FIT TO A SERIES OF USER-SUPPLIED DATA POINTS. IT IS CALLED AS
FOLLOWS:

        CALL ULSTSQ (X,Y,XN,COEFF)

WHERE X IS A USER-SUPPLIED ARRAY CONTAINING XN VALUES OF THE
INDEPENDENT VARIABLE; Y IS AN ARRAY OF XN VALUES FOR THE DEP-
ENDENT VARIABLE; XN IS THE NUMBER OF VALUES IN EACH OF THE X AND
Y ARRAYS; AND COEFF IS AN OUTPUT ARRAY WHICH CONTAINS THE N+1
COEFFICIENTS OF THE POLYNOMIAL WHICH WAS FITTED TO THE DATA.

  PUSH RETURN TO CONTINUE


IN ORDER TO SPECIFY THE DEGREE OF THE POLYNOMIAL WHICH IS TO
BE FITTED TO THE DATA, IT IS NECESSARY TO CALL UPSET BEFORE
'ULSTSQ' IS INVOKED.  FOR THE GENERALIZED CASE OUTLINED ABOVE,
A SUITABLE CALL TO UPSET WOULD BE:

        CALL UPSET ('POLYNOMIAL',FLOAT(N))

SEE EXAMPLE 8.10 FOR THE USE OF THIS CONCEPT.

  PUSH RETURN TO CONTINUE
```

OUTPUT FROM ALPHANUMERIC TERMINAL

FIGURE 7

```
GCS SESSION 6-6

CURVE FITTING
_____________

FOR THOSE USERS WHO DESIRE TO COMBINE DATA ANALYSIS THROUGH
CURVE FITTING INTO THEIR GRAPHICAL APPLICATIONS, SUBROUTINES
'ULINFT' AND 'ULSTSQ' SHOULD PROVE OF PARTICULAR IMPORTANCE.
'ULINFT' SHOULD BE USED WHEN IT IS DESIRED TO OBTAIN THE SLOPE(S)
AND Y-INTERCEPT (YI) OF THE LINEAR EQUATION,  Y = SX + YI.  THE
EQUATION REPRESENTS THE 'BEST' LINEAR FIT TO A NUMBER (XN) OF
USER-SUPPLIED DATA POINTS (CONTAINED IN ARRAYS X AND Y).
'ULINFT' IS CALLED BY:

         CALL ULINFT (X,Y,XN,S,YI)

LOOK AT EXAMPLE 6.8 PLEASE.

  PUSH RETURN TO CONTINUE AND THE EXAMPLE WILL BE PLOTTED
```



OUTPUT FROM TEKTRONIX 4014 TERMINAL
FIGURE 6

Acknowledgements

The above work was funded through the Army Integrated Software Research and Development (ISRAD) activity and the Office of the Chief of Engineers (OCE). The work was done at the R&D Software Group, Automatic Data Processing Center, WES, under the guidance of Dr. N. Radhakrishnan.

References

1. Graphics Standards Planning Committee, State of the Army Subcommittee, Graphics System Comparison Document, The Graphics Compatibility System (GCS) Software, June 1978.

2. Primer on Computer Graphics Programming for the WES ADP Center, Manual No. 78-1, August 1978.

3. GCS Programmer's Reference Manual for the WES ADP Center, Manual No. 78-2, August 1978.

4. Foley, J. D., Templeman, J., and Dastyar, D., Raster Extensions to GCS, September 1978.

Three-Dimensional Geometry Generation With A Practical Application

Fred T. Tracy

U.S. Army Engineer Waterways Experiment Station

The paper described techniques developed to generate both geometry and loading for structures. These techniques are applied to the practical application of analyzing stability of three-dimensional structures.

Geometry is generated by using three types of basic building blocks. They are: (1) two-dimensional cross-section with depth (called block), (2) 8 node brick element, and (3) group of planar polygon patches. The cross-section can be defined in the X-Y plane and extended in the Z direction or defined in the X-Z plane and extended in the Y direction. Line segments forming a cross-section can be either straight, circular, or quadratic, and any number of holes may exist in a cross-section.

Loading is defined in a consistent way as the actual geometry of the structure by defining geometries whose volume represent the amount of force. Directions (-X, +X, etc.) complete the definition of a force.

The three-dimensional stability analysis requires the computation of weight and centroid of the structure, and computation of forces and moments from the loads. Thus, algorithms were developed to do this.

THREE-DIMENSIONAL GEOMETRY GENERATION WITH A PRACTICAL APPLICATION

Fred T. Tracy
Waterways Experiment Station
Vicksburg, MS 39180

This paper describes techniques developed to generate both geometry and loading for structures. These techniques are applied to the practical application of analyzing stability of three-dimensional structures.

Geometry is generated by using three types of basic building blocks. They are (1) two-dimensional cross-section with depth (called block), (2) 8 node brick element, and (3) group of planar polygon patches. The cross-section can be defined in the X-Y plane and extended in the Z direction or defined in the X-Z plane and extended in the Y direction. Line segments forming a cross-section can be either straight, circular, or quadratic, and any number of holes may exist in a cross-section.

Loading is defined in a consistent way as the actual geometry of the structure by defining geometries whose volume represent the amount of force. Directions (-X, +X, etc) complete the definition of a force.

The three-dimensional stability analysis requires the computation of weight and centroid of the structure, and computation of forces and moments from the loads. Thus algorithms were developed to do this.

Introduction

These are several applications where three-dimensional (3-D) geometry must be generated and displayed. Further, volumetric computations such as volume, weight, centroid, moment of inertia, etc., are required. Thus more than points and lines must be specified. Questions also arise as to how to display the generated geometry. This paper describes techniques developed to generate and plot geometry and loading of 3-D structures. The practical application of 3-D stability analysis will be used as an illustration of the techniques.

Generating Geometry

Data that describe geometry are either points, curves, surfaces, or solids. Three types of solid pieces can be used:

a. Blocks
b. 8 node brick elements
c. Clusters of surface patches to form a solid

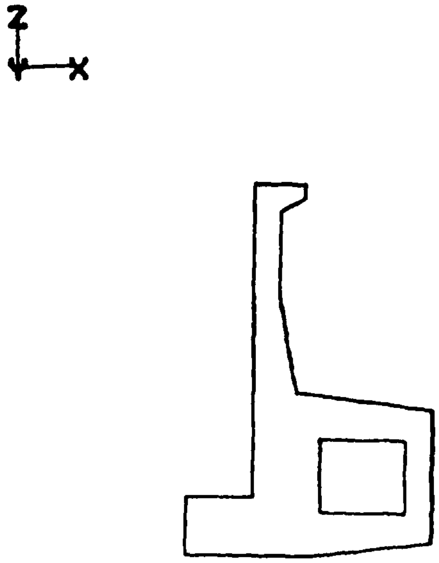


Figure 1 - Two-dimensional Cross-section

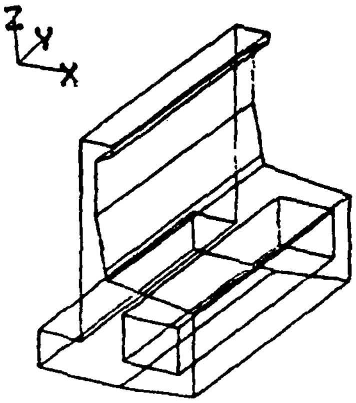


Figure 2 - Generated Block

Block

A block consists of a two-dimensional cross-sectional defined in either the X-Z or X-Y plane which grows in the Y or Z direction respectively to form a solid piece of geometry. Figure 1 shows a typical cross-section defined in the X-Z plane, and Figure 2 shows the generated block. The X-Z plane is the default plane.

Figure 3 with the accompanying data file shows two cylindrical blocks one generated from an XZ cross-section and the other from an XY cross-section.

```
10 XZ
20 POIN 4
30 1 -5 0 0
40 2 0 0 -5
50 3 5 0 0
60 4 0 0 5
70 CIRC 1 2 5
80 CIRC 2 3 5
90 CIRC 3 4 5
100 CIRC 4 1 5
110 BLOC BL1 100. 40.
120 1. 1.
130 4 1 2 3 4
140 XY
150 POIN 4
160 5 -5 20 5
170 6 0 15 5
180 7 5 20 5
190 8 0 25 5
200 CIRC 5 6 5
210 CIRC 6 7 5
220 CIRC 7 8 5
230 CIRC 8 5 5
240 BLOC BL2 100. 40.
250 1. 1.
260 4 5 6 7 8
```

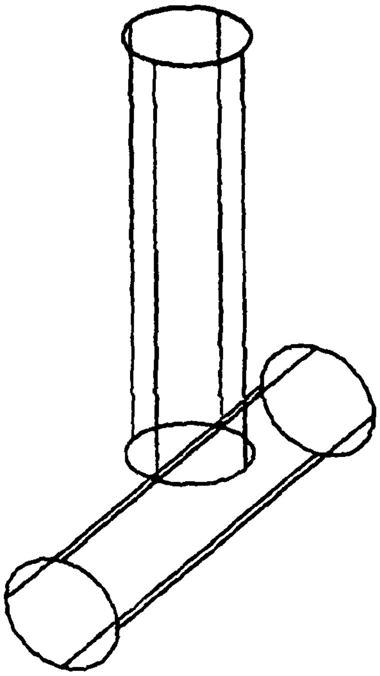
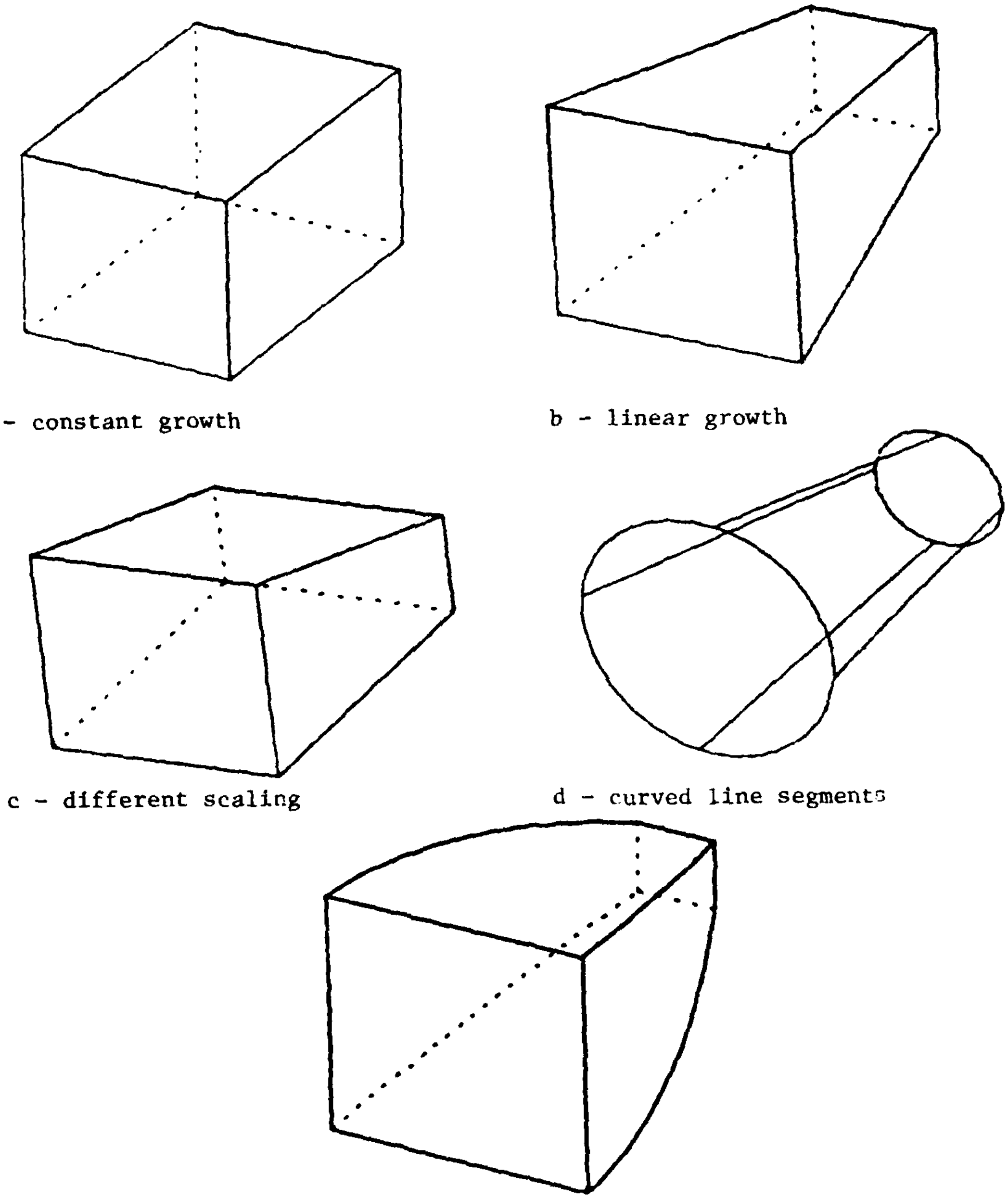


Figure 3 - Two Cylinders

Note that in the data file XZ says that the data refers to the X-Z plane, and XY is used to switch to referring to the X-Y plane. Also points are first defined, then any curves, and last the block itself.

The line segments describing the cross-section can be either straight, circular, or quadratic. Further, the section can grow smaller or larger as it is extended in the Y or Z direction. Any number of holes (culverts, etc.) can be defined in the cross-section as well.

The cross-section can grow in a constant, linear or quadratic manner in the third dimension to form a block. Figure 4a-e shows various types of blocks.



a - constant growth

b - linear growth

c - different scaling

d - curved line segments

e - quadratic growth

Figure 4 - Various Blocks

The great thing about the block is that the majority of data is generated with the user only providing cross-section information and a depth. The generated (X,Y,Z) coordinates are computed from the ones input for the cross-section (X-Z plane) by

$$\begin{aligned} \mathrm{YNEW} &= \mathrm{YOLD} + \mathrm{DEPTH} \\ \mathrm{XNEW} &= (\mathrm{XOLD} - \mathrm{XAPEX}) * \mathrm{SFX} + \mathrm{XAPEX} \\ \mathrm{ZNEW} &= (\mathrm{ZOLD} - \mathrm{ZAPEX}) * \mathrm{SFZ} + \mathrm{ZAPEX} \end{aligned}$$

where DEPTH is the depth, SFX is the scale factor in the X direction, SFZ is the scale factor in the Z direction, and (XAPEX, ZAPEX) are the coordinates of an apex. Interpolation points for the generated quadratic line segments are computed by

$$\begin{aligned} \mathrm{YINT} &= \mathrm{YOLD} + \mathrm{DEPTH} * .5 \\ \mathrm{XINT} &= (\mathrm{XOLD} - \mathrm{XAPEX}) * \mathrm{HFX} + \mathrm{XAPEX} \\ \mathrm{ZINT} &= (\mathrm{ZOLD} - \mathrm{ZAPEX}) * \mathrm{HFZ} + \mathrm{ZAPEX} \end{aligned}$$

where HFX and HFZ are scale factors provided by the user. Similar equations exist for a cross-section in the X-Y plane.

Brick

The 8 node brick element is another useful way to describe geometry. Higher order elements with curved sides will also be implemented.

Surface Patches

Sometimes it is desirable to describe a solid piece of geometry by a group of surface patches. This program allows planar polygon patches. Higher order parametrically defined curved surface patches will also be incorporated. Also, an interactive graphics program that allows input of 3-D geometry via the tablet (soon to be released) outputs patches.

Practical Examples of Geometry

Two examples of geometry are given here. They represent structures for Lock & Dam 2 on the Red River (work done by Mr. Tom Mudd and Mr. John Jobst, St. Louis District Corps of Engineers).

Operating houses on top of dam piers

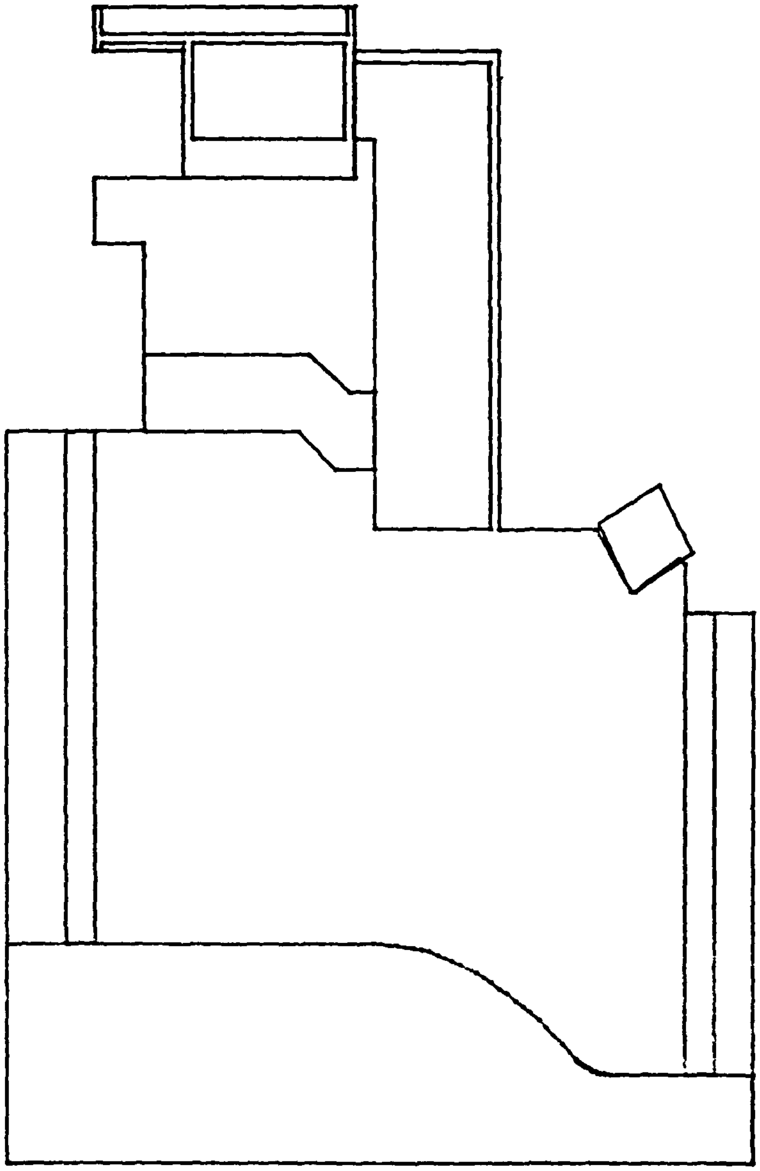


Figure 5 - Front View

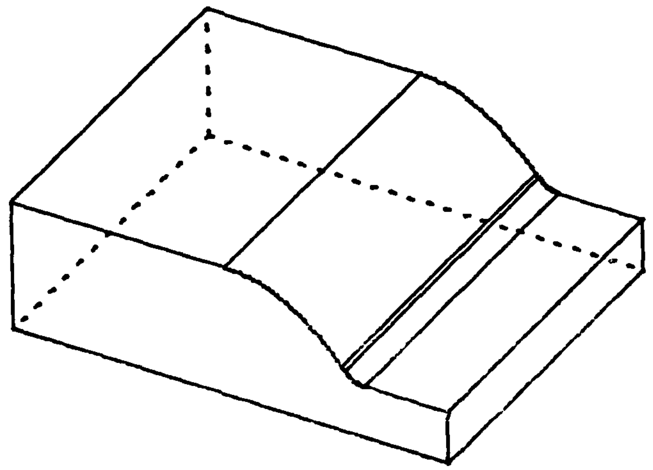


Figure 6 - First Block

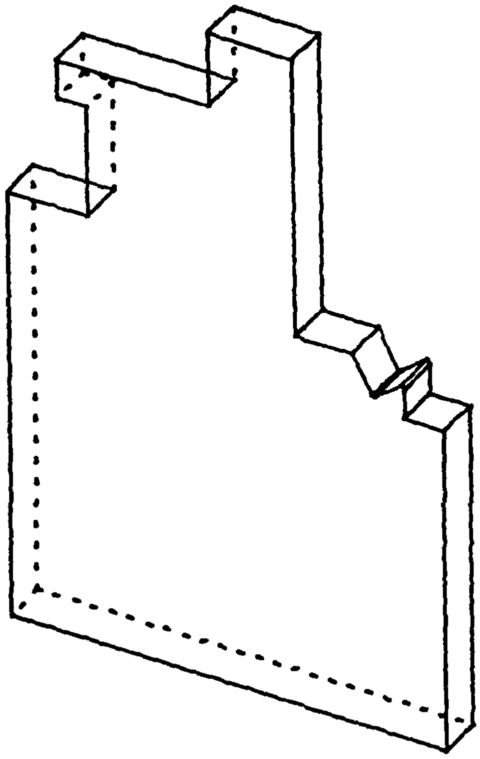


Figure 7 - Second Block

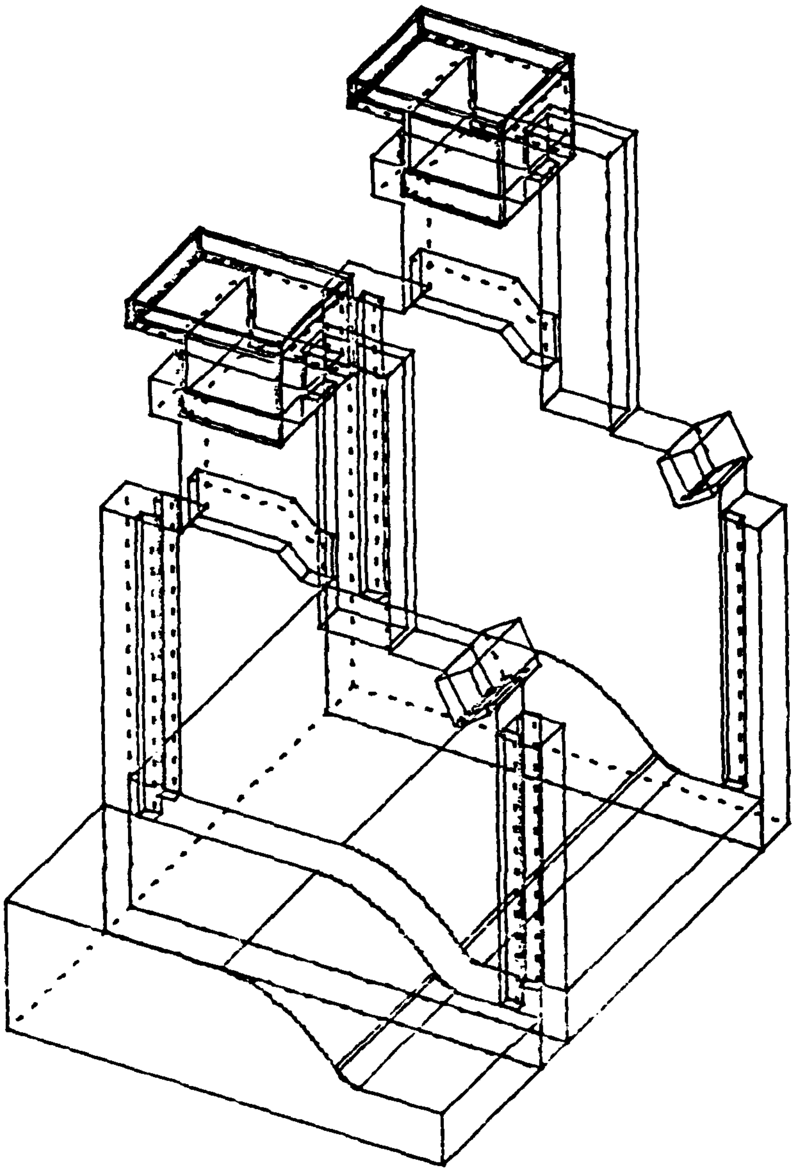


Figure 8 - Complete Model After Rotation

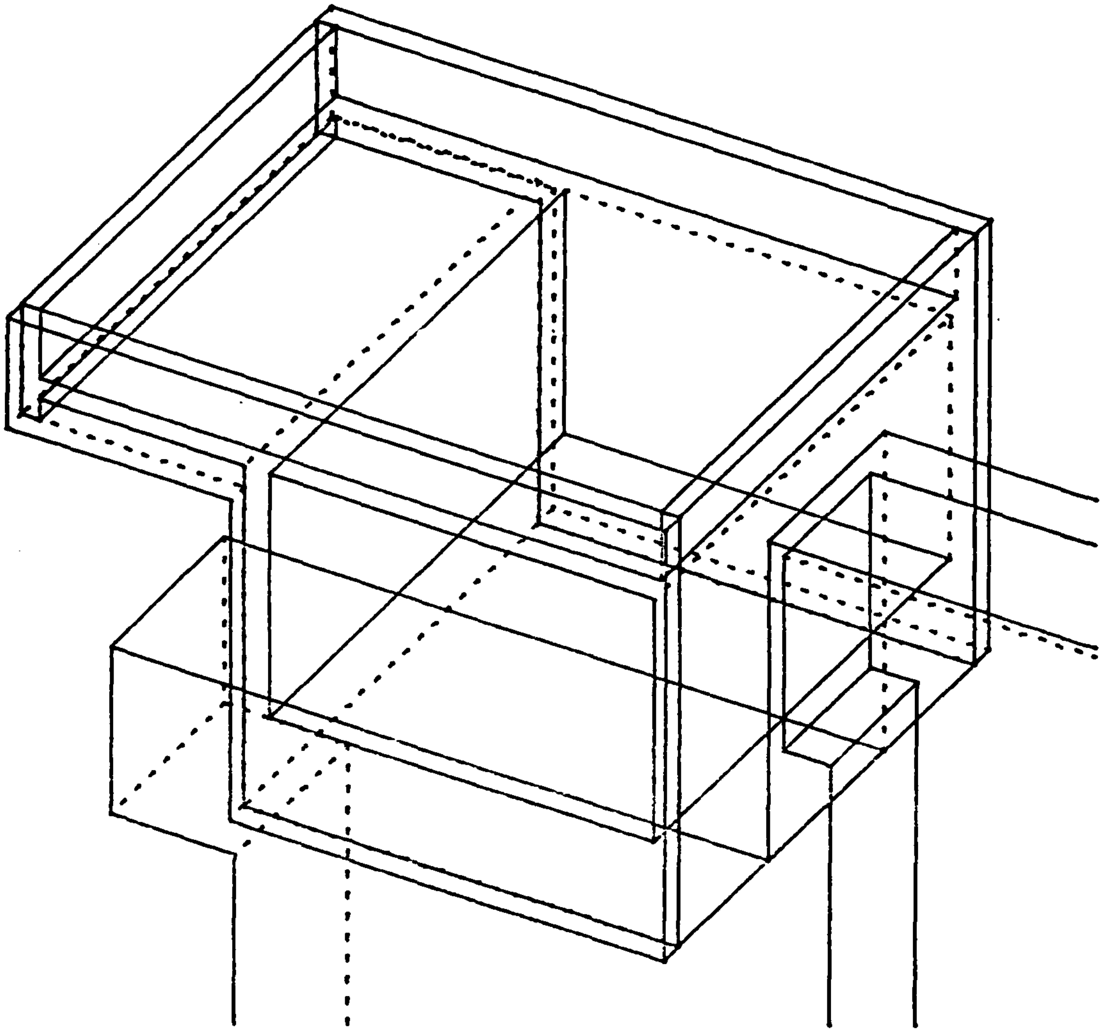


Figure 9 - Window Plot

Lock Monolith Near The Gate

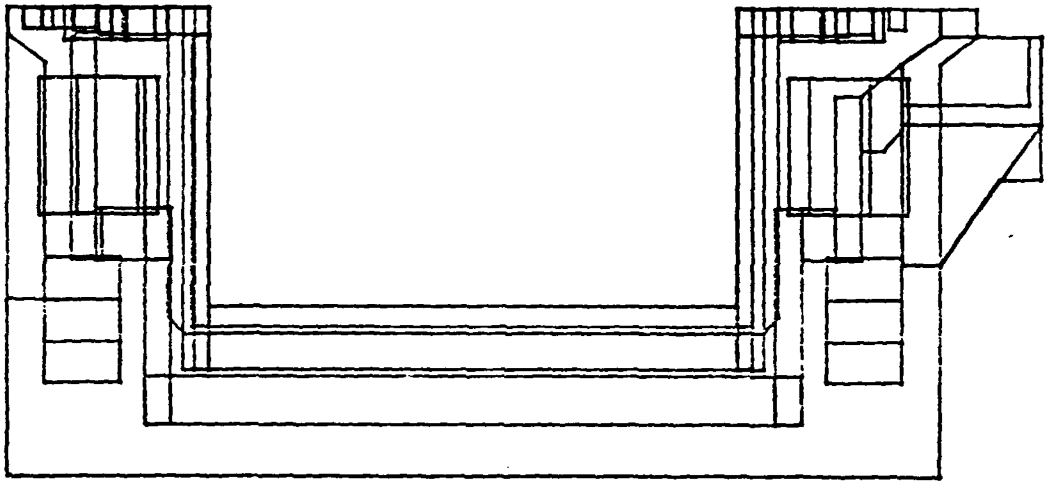


Figure 10 - Front View

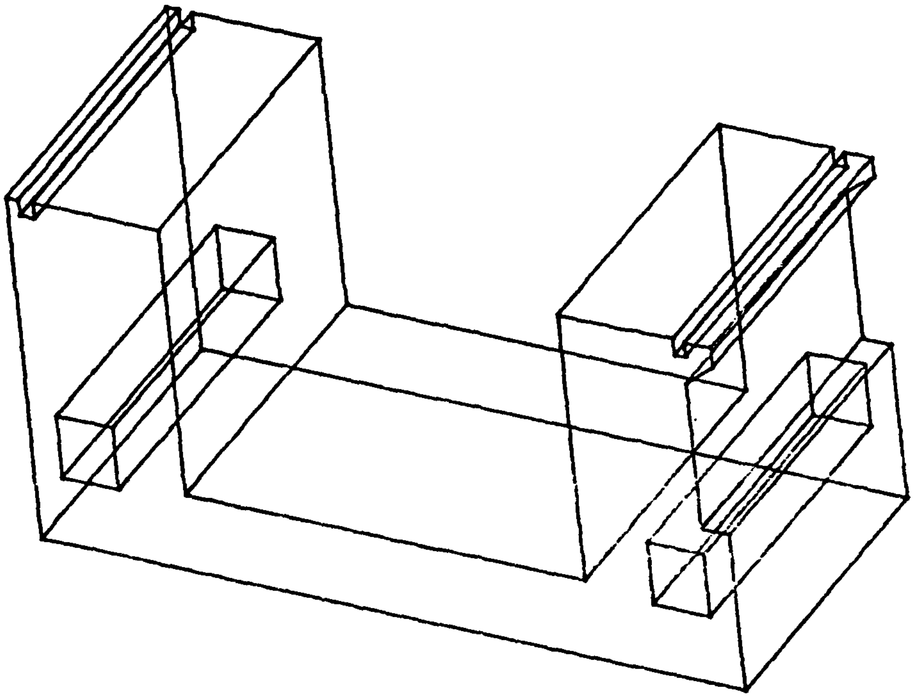


Figure 11 - First Block

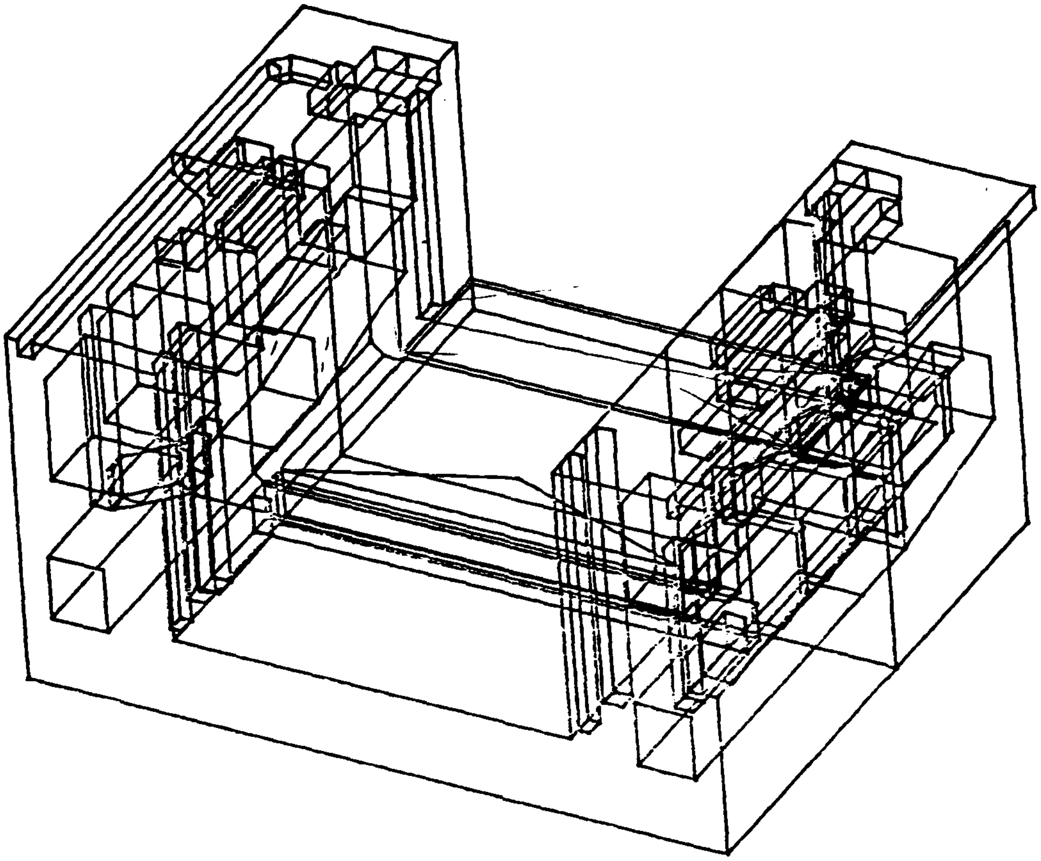


Figure 12 - Complete Model After Rotation

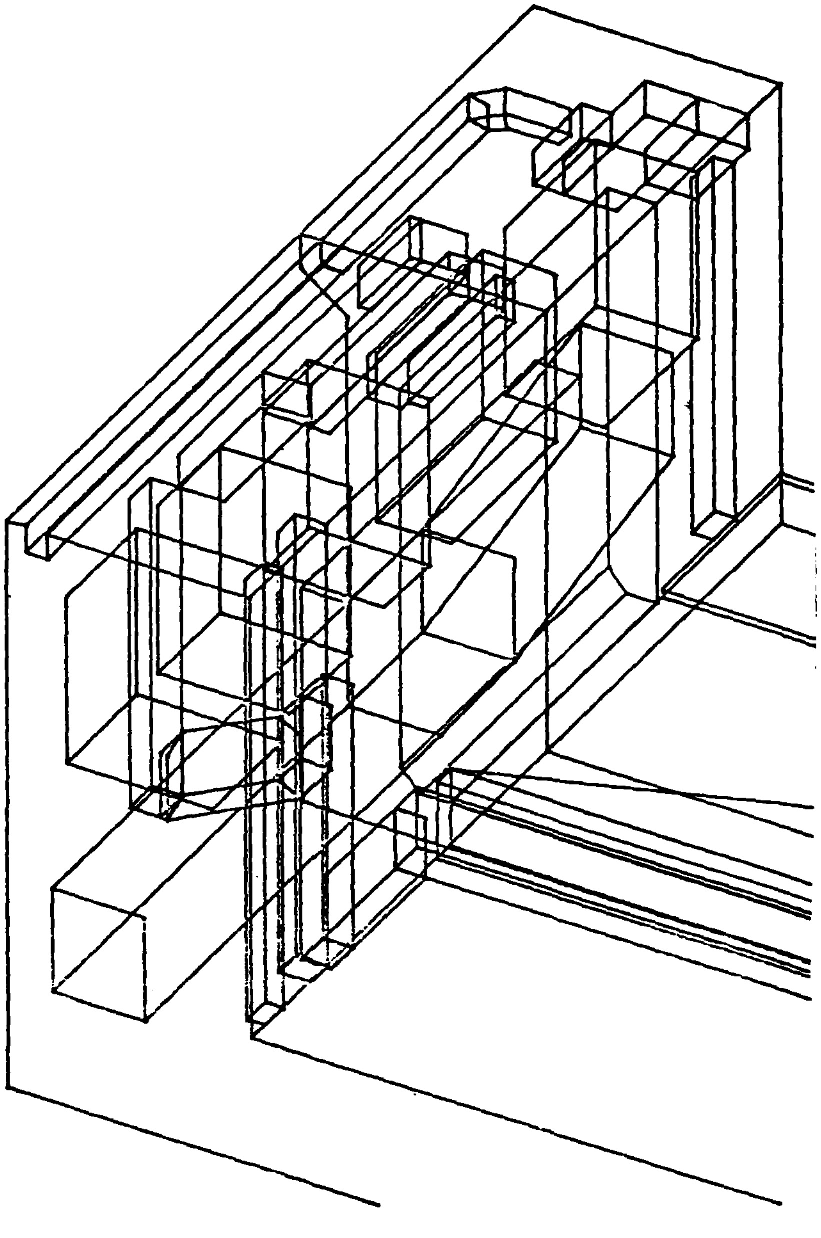


Figure 13 - Window Plot

Poor-Man's Hidden Line Algorithm

The "poor-man's" hidden line algorithm was used in the first example (Figure 5-9) to dash the hidden lines. Note that this worked well when plotting single pieces of geometry, but is not adequate for plotting the entire model. The second example (Figure 10-13) used all solid lines. This too is adequate only for plotting individual pieces of geometry. The conclusion is that full hidden line capability is sometimes required, but due to cost considerations, the poor-man's algorithm will suffice for editing purposes.

Generating Loads

The following general types of loads can be input:

a. Water
b. Unit strip
c. Weight
d. Line
e. Point
f. Planar piece

Loads are stored as geometries whose volumes represent forces. With a direction specified (-X, +X, -Z, +Z, etc.) the force is completely defined. A line load is stored as a surface whose area represents the magnitude of the force.

Plotting Loads

Two modes of displaying 3-D loads are required:

a. Display a designated cross-section of the geometry and loads.

b. Display in 3-D the volume representing the load.

Practical Example of Loads

Figure 14-18 illustrate a lock with its corresponding loads applied. The 2-D cross-section mode of display is used here.

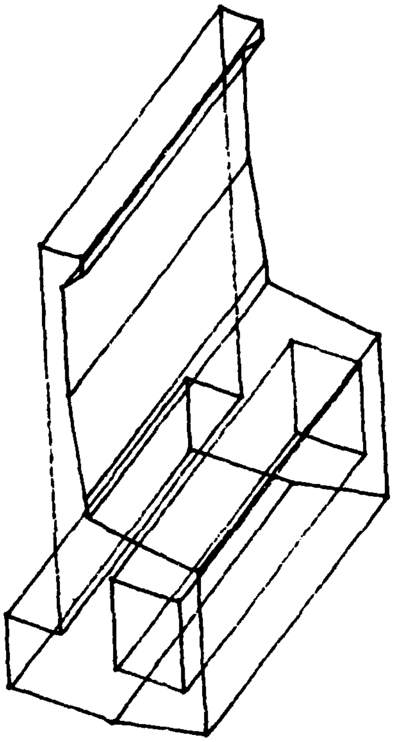


Figure 14 - Original Geometry

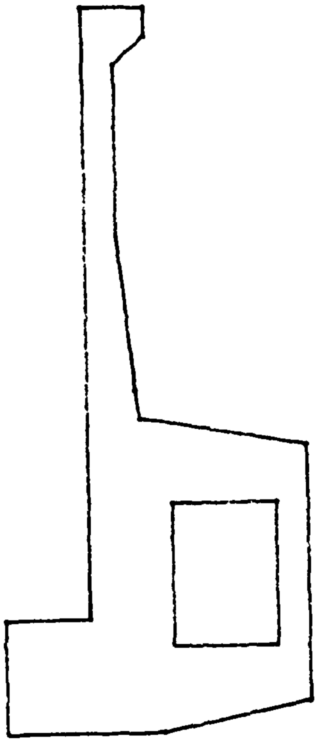


Figure 15 - Cross-Section of Geometry

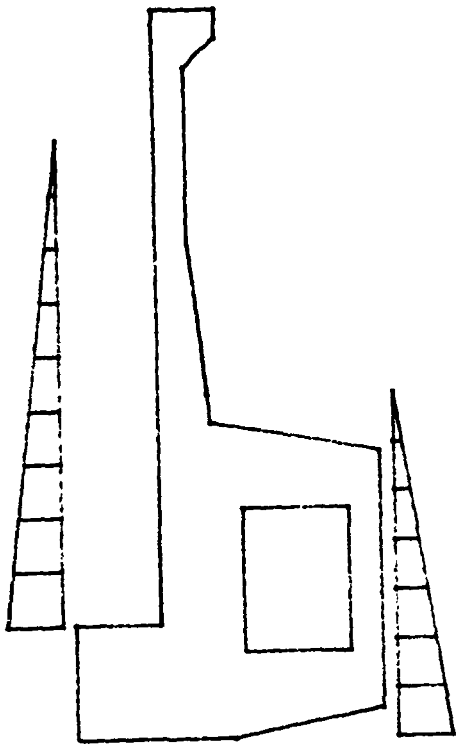


Figure 16 - Apply Horizontal Water Loads

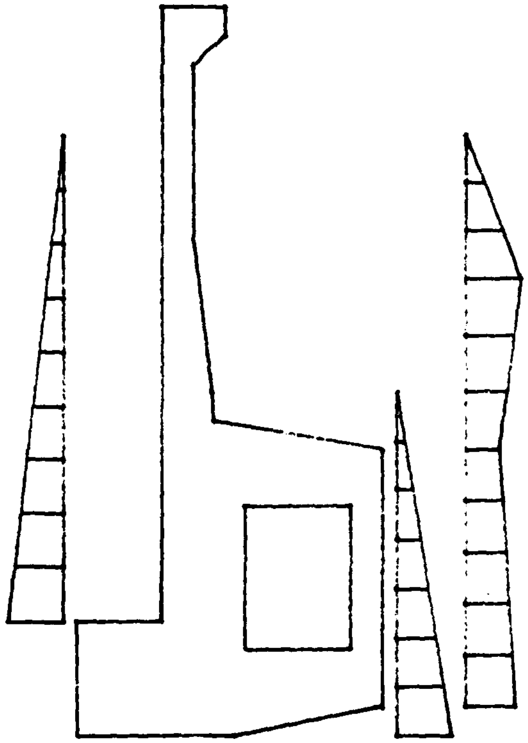


Figure 17 - Apply Horizontal Soil Loads using STRIP Command

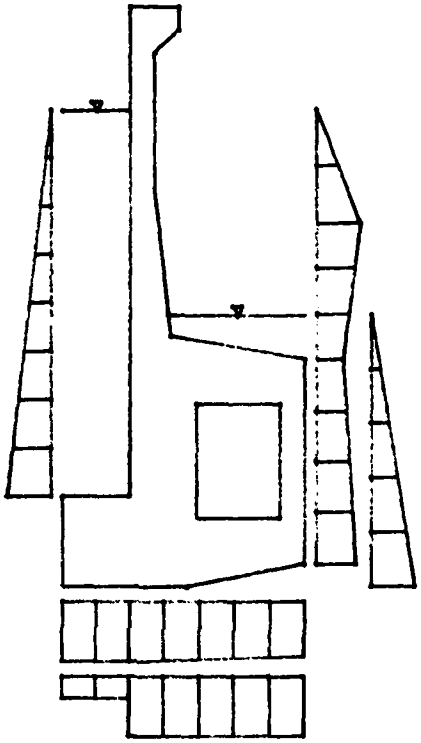


Figure 18 - Apply Vertical Water and uplift Loads

Analysis

From the general geometry module volumes, weights, and centers of gravity are computed. From the general load module forces and moments in the X,Y, and Z directions are computed. This information can be applied to the base of the structure to determine its stability.

Acknowledgements

Special thanks go to Dr. N. Radhakrishnan, Special Technical Assistant, ADPC, WES, for his advice and support in this project.

Coherence Concepts in Computer Synthesized Real-Time Displays

John Staudhammer*

U.S. Army Research Office

*(On Leave from North Carolina State University)

A computer display device and supporting software is described for the display of images of three dimensional objects directly from a minicomputer to studio quality television. The system is capable of displaying complicated moving objects. Each frame of a television picture is generated using a compact image run-length code so that each frame may be expanded to a normal television signal in 33 milliseconds. Sequences of these images are used to generate real-time cartoons without the use of photography.

The success of this compaction scheme depends on the existence of coherence in the synthesized image. Normal television scenery contains five details in the background to make this scheme successful. However, synthetic scenes, such as are used in computer aided design of object assemblies, generally are preferred not to have busy backgrounds. In these cases the encoding scheme can be used with great success. The particular hardware described here is designed to operate on the PDP-11 Unibus using a normal DMA controller which limits the usable data rate to 800,000 bytes per second. Thus, each TV frame must be described with about 26,000 bytes of data.

Objects having several hundred facets, with or without smooth shading, and objects having curbed surfaces, such as space filling atomic models, can be successfully displayed. All these objects show coherence such that from a visible image point a simple prediction can be made of succeeding visible points. Special digital hardware operating at television frequencies is used to generate groups of display points.

COHERENCE CONCEPTS IN COMPUTER SYNTHESIZED REAL-TIME DISPLAYS

JOHN STAUDHAMMER

U.S. Army Research Office
Research Triangle Park, North Carolina
(On leave from North Carolina State University)

ABSTRACT

Details are presented of a system using special hardware for displaying views of three-dimensional objects as a succession of color television images of studio quality in real-time. This cartoon film creation capability operates from a conventional minicomputer and uses a special video generator. The generator receives control information from the computer, at normal computer speeds, and from this generates real-time video signals. Visibility and shading are pre-calculated by the computer, but are played back in real-time and may be recorded as a standard video tape. Objects having hundreds of planar polygonal facets and smooth surfaced objects having several hundred curved surfaces may be displayed as an animated cartoon.

INTRODUCTION

Three-dimensional objects, such as architectural assemblies, cartographic surfaces and sculpted hulls are usually described by an ordered set of points for display by computer graphic devices. The simplest display processors will draw a straight line between successive data points thus producing a lofted or ruled surface representation or by merely drawing the edges of the planar polygonal surfaces that delimit the object. Sometimes a smoothing process is applied to produce many plotting points which lie on some interpolating curve, often a spline function, controlled by the surface data points. In this latter case too the plotted points are still connected by straight lines. Each line in the image can be considered an edge where two surface elements meet. Until there is a very large number of such edges in a small area of the image the surface will have breaks and will not appear smooth.

For visualization of the surface mechanical plotters and direct-view cathode ray devices are the devices used predominatly. An object represented by such line drawings is termed a "wire frame" display. For interactive graphics usage hardware can be used to rotate, translate and scale

This work was supported by Grant MCS75-06599 from the National Science Foundation, Division of Computer Systems Design.

such wire frame displays under the immediate control of the operator. Objects having several thousand edges may be so manipulated by well-established hardware manufactured by such vendors as Adage, Vector General, and Evans and Sutherland. Figs. 2 and 9 show such wire frame pictures. Note that a sense of mass is missing from these objects: their images appear as chimerical phantasmagoria.

A more ready visualization of the objects is achieved by the use of a half-tone rendering of the visible parts. Much effort has been expended by various groups of workers over the last ten years to achieve ever better looking imagery in computer produced photography. For interactive graphics, for example in the design of buildings and vehicles, photographic imaging is typically too costly and too slow. Unless images are produced relatively rapidly, typically in a fraction of a second, the thought process in the design activity is disturbed to such a degree as to interfere with creativity. Photography is, however, the prefered medium of documentation of the (static) results.

By far the most cost-effective process for visualization of complicated three-dimensional constructs is through the use of color television. Television may be used to present a large number of views of an object being designed to an observer. While the image resolution of an individual view may be limited, the collection of views, the cartoon film, can present much more information, so that a user may learn much about the designed object. For example relative motion of the various parts of the object during rotation and translation will give vivid clues to shape, size and depth.

The use of standard or near-standard television allows the computer graphics user to benefit from the development efforts in the TV industry, which produced economic color display devices of acceptable resolution. The broadcast TV standard image consists of 525 lines, 485 of which are visible. A new set of such lines is painted once every 1/30 second. Each line is made up of about 52 us of visible scan (from left to right on the normal TV display) and 12 us of blanked retract (from right edge to left edge.) Hence the vertical image sampling is 485 lines. Each line can be thought of as being made up of a series of colored dots, the picture elements, or pixels. Since the normal display has an aspect ratio of 3:4 (vertical to horizontal size) one would require about 648 pixels to have a TV image equi-distantly sampled in both the horizontal and vertical directions. For such an image the inter-pixel timing would be about 80 nanoseconds.

The usual practice is to have a buffer memory which is continually read with a video rate counter. The buffer memory is an exact image (in digital form) of the display screen contents. Usually one byte of grey level (256 levels) is allowed for black/white displays. If all three channels of a color display were encoded with eight bits each there is a requirement for 24 bits for each pixel. This scheme would allow over 16 million different colors. Seldom is such wide choice required, except in special discrimination studies. Even natural scenes contain far less color, at least in major sections: skies are mostly blue (or brown, depending on geography), lawns are green (or brown), buildings have near-uniform color, etc. Consequently far fewer bits are required to describe a scene and a color lookup table (a "pallette") is used to read the true color represented by the stored information. In essence, the display image becomes a large "paint by numbers" creation, with the numbers possibly having various meanings in different parts of the image. Normally a one-byte descriptor is used in the buffer memory with a "mode" indication as to which of several pallettes are to be used.

Some commercial display devices operate with 480 lines and 640 pixels/line, close to the above specifications. A buffer memory containing 512 lines x 640 pixels is easily constructed from five 64 K memories. Using today's technology buffer memories can be constructed with 5 memory chips/bit in the buffer memory. Display lines 481 through 511 can be used for control storage or the display could be scrolled. Other display systems don't use the full horizontal scan and produce a 512 line by 512 pixel image or, using the full horizontal scan, use a 3:4 pixel spacing on a 512 x 512 grid. Most display devices fill the display buffer from a standard computer and therefore take a rather long time to change an image. At a one Megabyte I/O transfer rate from the computer, the required time of one quarter seconds is about ten times the frame repetition rate of normal television. Since most minicomputers have lower I/O rates and require additional protocol time to complete the I/O transfers, buffer fill times in excess of one second are not uncommon for systems using normal computer I/O channels for image subsystem communications. Such images are clearly not suitable to dynamic cartooning.

Even if a multiported memory is used, where fill time for the buffer can occur at CPU speeds, the required 100 ns inter-pixed time for a 512 x 512 image requires a very fast computer and extensive parallel operations in that memory. Additionally, digital storage of sequences of images requires a vast capacity. For a one-minute cartoon sequence of 512 x 512 images with a one-byte color pallette encoding there is needed a storage capacity about 450 million bytes.

These formidable requirements clearly require compromises in design and capabilities. Basically the computer output rate is about an order of magnitude less than the display data rate required by a television display. Additionally the data storage capability of minicomputers in main memory is but a fraction of a single image. A digital disk memory subsystem has much more capacity, but the data transfer rate is not high enough to sustain a pixel by pixel video image description. While it is true that very large computer systems do have adequate I/O rates and storage capacities, economic designs for minicompters have to achieve significant reductions in both data rates and in the amount of required data. The scheme described here does both.

RUN-LENGTH CODING

For use in design of shapes and in data representation, spatial relations between elements of a scene must be depicted. Usually the elements are relatively simple, and they appear on simple backgrounds. The illustrations in this paper are such examples. As an example the image pairs in Figs. 8 and 9 are of the same design object (a "space ship"). It is evident that the background is simple (it is a uniform "black") and that any one scan line, a horizontal line through the image, contains but a few changes in color. Since the object is made up of planar polygons, each visible polygon part will have the same color throughout; particularly the visible segment of such a polygon on a given scan line will be a constant color. Hence each scan line can be encoded by a simple sequence of (color, count) combinations. We refer to such an encoding scheme as run-length encoding.

The color that may be represented in this system is a combination of 32 intensity levels of Red, Green and Blue. This allows some 32,000 different colors in the display device and requires 15 bits per color. Since any one image may be made up of only a much smaller number of colors, we may consider a "paint by number" scheme involving encoded color values. If we restrict the number of colors to 256, the colors themselves can be stored in a "palette" and referred to by a color-number (of 8 bits). Note that the color gradations themselves are limited by the palette depth and could conveniently contain 8 or more bits per primary color. Additional flexibility may be obtained by using several palettes or an offset in a larger palette, selected dynamically during image generation.

Similarly most images will be made up of some long bars of constant color (such as backgrounds) and many short constant-color segments. The count words will be mostly small integers requiring but a few bits; however, design simplicity usually dictates a pre-set number of bits for the count even if often the leading bits are zeros. An 8 bit count is a convenient compromise.

Hence the run-length coded image will basically be a collection (count, color) two-byte pairs of words. Provisions are also made for special commands such as line start, line fill, field start, palette select and mode select (as when one wants to display single dots only or to rewrite palette locations.) These command words can be conveniently triggered by a ""run-length" command of zero followed by a one-byte instruction.

The generation of the image then becomes an interpretation of this data structure. Since the pixel rate for a 512 x 512 image is about 100 ns per element (it is about 85 ns for a 640 pixel line displayed at "normal" TV format) the image generator must be a digital device running at about 10 MHZ rate. Using Schottky TTL technology this is achievable.

The display generator should contain a buffer which holds the display commands. If this generator is to produce moving images, one needs to update the buffer every 1/30 second for normal, broadcoast-compatible TV. The size of the image buffer is therefore limited to what can be transferred to it in 33 ms. For one of these generators to be operating on a standard Unibus, the transfer rate, using a standard DR-11B DMA controller, is 2.5 microseconds per 16 bit word. Thus the maximum buffer size needed for images updated every 33 ms is 13,200 words (i.e. 26.4 K bytes). If a double-buffering technique is used where the image generation using one buffer is overlapped with the update of the other buffer, large display lists may be handled. Normally, however, if images are updated less than 15 times a second, annoying "jerkiness" results in the perceived (non-flickering) image sequence.

SOME HARDWARE DETAILS

A single buffer system is shown in Fig. 7. This unit measures 8x8 x 17 inches and is driven from a standard PDP11-45 Unibus through a DR-11B DMA controller. This device is an improved version of an earlier computer controlled generator[1] which was used to produce the image shown in Fig 8. Both the older and the newer generators produce studio quality TV imagery. The newer version was custom built by this author and his associates and is in operation at Ohio State University. The images in Figs. 2-6 and Fig. 10 were generated by this device, each image in 33 ms, and displayed on a normal color TV receiver. Sequences of such images can be recorded directly on a TV recorder thereby creating a videotape of a color-TV program without the use of photography. Several tapes made by this method are listed in the references, in Section B.

The video generator consists basically of two first-in first-out (FIFO) stacks. One, consisting of 16 K 16 bit words is the buffer referred to above. It can be filled at the Unibus transfer rate through an interface to the DR-11B in the computer system. This first FIFO can simultaneously be emptied into a second FIFO of about 2 K bytes on demand from the second FIFO, which has a cycle time equal to the TV image pixel rate. This fast FIFO is used to buffer busy segments of the image, which may be displayed with each point possibly requiring a separate instruction. Since the large buffer may be filling with the next image list when a given image command is required by the TV generator, the function of the fast buffer, the "Detail Buffer (DB)" is to load level between the image output rate and the "Image Buffer (IB)" fill rate (determined by the computer output rate). A controller manages these two buffers and also issues requests to the computer to put data on the DMA output lines.

Within the computer system sequences of output images ("frames") are kept on a large fast disk, a 44 Mbyte RJP04 disk drive. During image generation, the controller of the TV generator seizes the Unibus, effectively usurping all the system resources for the image generation.

Besides the two FIFOs the TV generator contains table look-up hardware for the palette and a set of D/A converters for generating Red-Green-Blue primaries which are then displayed on a RGB monitor and may be further used to derive baseband video or modulated video (channel 6) for CTV distribution. The baseband video may be recorded on a normal commercial TV recorder, thereby producing a TV film. Examples of imagery produced by this system are Figs. 2-6 and Fig. 10.

SOME SOFTWARE DETAILS

Much software was generated for this system by the Computer Graphics Research Group at Ohio State, Dr. C. Csuri, Director for making the device conveniently usable in interactive cartoon generation. This work is described in (3, 5-7).

In composing a cartoon sequence, the operator constructs a starting frame and specifies motion of parts of that frame (or specifies an ending frame). Between key frames all intervening frames are calculated. The composition of each key frame and of motion of the key frames is done interactive by using a wire-frame display of objects (see Figs. 2 and 9). The operator sits at a console containing two display screens, as shown in Fig. 1. On the right the wire frame display appears, on the left a shaded color image is shown (examples are Figs. 3 and 4). The shaded image is calculated in background mode and displayed with a periodic update. The time between these updates depends on the image complexity and the computing power available to service the displays. For an image complexity represented by Fig. 3, the visibility of surface elements, their shading and the conversion of the visible scene to run-length encoded form can be accomplished in under one second (Ref. T3). Construction of a key frame proceeds from retrieving primitives from a disk file (i.e. "cubes"), molding them interactively (i.e. moving corners under cursor control), assembling a collage of such objects and assigning certain attributes (i.e. color for faces, subassemblies, etc.). Direct view of the object is provided by the vector display, a Vector General 3D display. Each object is made up of a collection of opague polygonal facets. Each vertex point is described by a triplet of (x, y, z) coordinates, each value with a full 16 bit word. Hence a rather large dynamic range of scaling is available before data accuracy problems appear as granularity of the produced image. This system is therefore a true three-dimensional "cartooning" system. Indeed objects can be made up of intersecting surfaces, surfaces with discontinuous edges (i.e. cutouts, such as windows in a wall) and objects can be made to traverse each other.

Dynamic effects on the key frames, such as zoom, pan, motion and rotation, are handled by a software package which allows the operator to specify various effects which are simultaneously applied to the key frame to produce subsequent frames, and thus the illusion of motion. Calculation of the required run-length commands is a software function and normally requires off-line calculation. The run-length commands for a display sequence are stored on a disk; at the end of the calculations for all frames in the dynamic sequence, the TV generator hardware described above is turned on. This hardware then "dumps" the disk content to the scan display.

Editing of sequences can be done rapidly. Since software is used to control virtually all facets of image generation, the resolution of calculations need not be identical to the attainable display resolution. Rather one could calculate a cruder image (say on a 256 x 256 raster spacing) to check on image dynamics, coloring, sequencing, etc. Only when a satisfactory design has been achieved, would one need to generate all display points for a pretty image.

Similarly, if a photographic quality image is desired, assuming the availability of a photographic display device, the same data structure can be used to calculate a higher resolution image, and produce the required commands for photo production. The advantage of the video display mode is that it can be used for interactive creation of sequences, which are impossible with photography due to the long turn-around time required by photographic processing. We note in passing that the Polaroid instant movies do not have the spatial resolution, the signal/noise ratio, the color fidelity and reproducibility of the studio quality TV imagery possible in this system.

SOME APPLICATIONS

A system comprising the hardware and software items outlined above is described in (3). The system has been used to produce a number of video tapes running as much as an hour. A 44 Mbyte disk has enough capacity to hold about one minute's worth of run-length commands; sequences of such short animations are edited as normal television tapes to yield a full show.

The subject matter can be almost anything. Tapes have been produced to describe the system (T1), to explain the creation of animation sequences (T2), to illustrate geographic surfaces and population trends (T3), to produce teaching aids for deaf children and several other subjects. Several single frames from (T3) are shown in Figs. 1-5.

Since the system is capable of creating images of objects containing several hundred thousand edges, images of curved objects can be produced without noticeable faceting. Similarly surface texture can be approximated by many small facets as is shown in Fig. 6.

EXTENSIONS

Three dimensional moving objects form the basis of the dynamic imagery produced. Part of the software process is the calculation of apparent color from a knowledge of the true color of a surface and the orientation of that surface relative to the observer. Consider a cube which has faces colored the same viewed from a direction along one of the body diagonals. The outline of the visible body is a hexagon and if all visible surfaces were colored their true color, a hexagonal flat surface would appear on the display screen. In order for the edges between faces to be perceived, there must be a slight color change at the edges. This is accomplished by modifying the "true" color, i.e. the color one would see when looking normally at the surface. The modification is a function of the angle formed by surface normal with the direction of the observer. Thus for each planar polygonal surface a separate modification function must be calculated. However, only one such calculation need be done for each surface, no matter how many visible points it possesses; also the function used is some power of the cosine of this angle, and therefore relatively easy to calculate.

More sophisticated coloring algorithms may consider distance to the observer of a visible point in assigning a value to the apparent color. Such calculations become very time consuming, but can simulate more interesting visual effects, such as fog and clouds.

One may also use surface coloring as an indicator of an other dimension. For example a complex function of a complex variable has really four dimensions: the real and imaginary parts of both the independent and the dependent variables. Alternately the dependent function may be represented as a magnitude and a phase angle, consistent with such usage in the automatic controls literature. On the simplest level the independent variable is chosen as the x-y plane, the magnitude of the dependent variable as the z-direction and the phase angle of the dependent variable as a color scale. The relations are:

$$s = x + jy$$

$$F(s) = F(x + jy) = P(x, y) + j\,Q(x, y) = M\,e^{j\,\theta} = M\,\underline{/\theta}$$

$$M = \sqrt{P^2 + Q^2}$$

$$\theta = \text{arc tan}\ (Q/P) \qquad -\pi < \theta \leq \pi$$

The variables (x, y, M) are determined as a triplet of values and displayed as an (x, y, z) 3-D value. The surface is generated on a grid of (x, y) values in a limited range of (x, y):

$$\text{XLOW} \leq x \leq \text{XHI} \qquad x_i = x_{i-1} + \Delta x$$

$$\text{YLOW} \leq y \leq \text{YHI} \qquad y_i = y_{i-1} + \Delta y$$

and the corresponding M_i, θ_i values are determined. The surface is then made up of triangular surface elements whose vertices are neighboring points. The surface is thus a set of triangular tiles, whose color however is assigned a value representing the average of the three angles at the three vertices.

The color scale itself must be chosen carefully. We note that the angle θ may be a full circle value; therefore, the color representation must be a scale which unambiguously runs through a set of values and closes back on itself. Such a circular scale cannot be achieved with a binary representation, such as a gray scale. However, the three primary colors Red-Green-Blue may be so used. We also note that the phase angle scale should have a large number of gradations (i.e. should be of five granularity) so that a good representation can be achieved. Fig. 11 shows such a surface calculated on a 41 x 51 (x, y) grid; moreover, the magnitude value, shown vertically on the figure, is the log magnitude M':

$$M' = \ln (M) = 1/2 \ln (P^2 + Q^2)$$

(We note that calculation of the logarithm or the square root requires basically the same amount of time). The color scale is made up of 96 elements:

$$0^o = (\text{Red} = 0, \text{Green} = 32, \text{Blue} = 0)$$

$$120^o = (\text{Red} = 32, \text{Green} = 0, \text{Blue} = 0)$$

$$-120^o = (\text{Red} = 0, \text{Green} = 0, \text{Blue} = 32)$$

This arrangement allows a fineness of $360/96 \approx 4^o$ per color value; in any given range for each angle increment one of the primary colors is decreased linearly while the other is increased. For example in the range -120^o to 0^o for each 4^o increment in the angle the Blue value is decreased by one and the Green value is increased by one. This results in a scale of binary colors; one might replace any one of these with a tertiary color, such as White, to identify a given phase angle without additional comp"tations.

Similar schemes can be used to introduce one higher dimension by using time as another independent representation. Also one can use two colors as representations of two independent variables; however, there are not very many colors that can be distinguished as absolute values by eye, hence only a crude representation is possible. By using a 3-D grid, 2 different color scales and time, each as orthogonal scales, one can represent six dimensional functions in this manner (9). It must be pointed out that sequences are complex variable surfaces, in essence a five dimensional function, is about the limit of reasonable representation in this scheme.

Clearly great simplifications in the above display processes are possible if one were to work only with two-dimensional patterns. A system for pattern design ("textile patterns") was demonstrated using the display generation schemes described in this paper which allows interactive creation of such patterns.

SMOOTH SURFACES

The representation of a 3-D object by a limited number of planar polygons can lead to a faceted representation of curved surfaces. For example a smooth sphere will have the appearance of a golf ball; see also

Figure 8. One gets away from this faceting with planar polygonal representations only if very small color changes occur between adjacent polygons. Usually this is achieved by using a very large number of polygons in the object representation; hundreds of thousands of edges may be used if one wants a high quality image. Adding texture to the surface helps also; but the process of texturing may introduce many edges in the imagery.

One may handle a limited number of polygons in the basic object description and use a finer representation only for the visible parts of the object. This scheme reduces the visibility calculations required but will leave boundaries (i.e. silhouettes) as a set of relatively crude straight lines. Again special processing may be used to improve the silhouettes' fidelity.

The advantage in handling planar polygons for the object representation is that all the algorithms are well known, many have been reduced to special hardware, and many of the processes have been structured for efficiency of display.

Another method is to represent the objects with better approximation primitives. For example the spaceship "engine" in Figs. 7 and 8 could be made up of a cylinder having curved sides rather than a set of planar rectangles. In general the polygonal surfaces are made up of a collection of rectangles, that is, areas where the position of each boundary point of the region is prescribed by the straight edges. (Unless special care is exercised, these regions are not planar; however, one can always approximate each rectangular region by two triangles which are guaranteed to be planar.) This representation insures surface continuity on the object, but the abrupt slope changes across the polygon edges lead to a coloring change. Since the eye is a very good image differentiator, these edges become enhanced and lead to the apparent faceting in the display.

To preserve continuity of the slope across neighboring surface areas ("patches") one needs to prescribe not only the function value, but also the slope of the patch at the boundary. For rectangular areas two local orthogonal parametric coordinates may be used. Usually these coordinates are normalized to the range 0 to 1. Then the surface description will be two orthogonal families of functions. The value and slope of every member function are prescribed at two points. The simplest of such function is a cubic, and the representation becomes a bicubic description of surface patches.

In generating visible points from such surface descriptions, visibility calculations become more involved, the surface normal must be calculated at each visible point separately, but silhouettes are no longer approximated by straight lines. The number of surface primitives (i.e. patches) is far less than with polygonal approximations giving comparable fidelity. A study of some simple objects (a bottle) rendered by facets (over 300) and bicubic patches indicated a reduction of an order of magnitude (8). The surface normal calculation requires the lion's share of the total computing; however, some faster approximations may be developed in the near future to speed this task.

This process can also handle translucent objects, such as artificial images of transparent glassware. The bicubic object description/image generation scheme is general in that arbitrary objects may be represented. Specialized hardware must be designed to make this scheme operate in near-real time (8).

Another extension of this display generation scheme is the production of specialized object images in real time. Prism-maps, such as the one shown in Fig. 5, can be produced with special hardware. Here advantage is taken of the image structure coherence in that all surfaces are normal or parallel to another. Indeed only one angle need to be calculated for the entire set of prism tops, and this angle is modified with a simple calculation to determine orientation and a-priori visibility of all surfaces. Only obscuring of some surfaces must be calculated; this is a much simpler task than the random surface orientations in Figs. 3, 4, 8, 10. Real time display of images of the complexity shown in Fig. 5 may be achieved with modest hardware.

Specialized curved surfaces may also be easy to represent with modest equipment. For example molecular models require spherical surfaces. Fortunately the spherical surface is the implest of doubly curved surfaces to handle (without perspectives). Every section through the surface has a circular trace, the shading of which may be kept in a fast read-only memory. All that is required is to scale the ROM function to the proper size in the display, and to assign the basic color for each section of each sphere. Visibility calculations basically reduce to finding the starting address in the ROM where the shading function will begin to be read from, the address increments to take in the ROM to achieve proper scaling and the number of points to read. Together with coloring information all the required data can be packed into about four bytes and decoded by special hardware to achieve a point-by-point differently colored spherical molecular display of fairly complicated molecules (10). Two very simple molecule models are shown in Figs. 11 and 12. The calculation of visibility is far more difficult, but special algorithms may eventually make even this type of display possible to operate in real time with only modest hardware.

SUMMARY

A computer display device and supporting software was described for the display of images of three-dimensional objects directly from a minicomputer to studio quality television. The system is capable of imaging complicated moving objects. Each frame of the television picture is generated using a compact image run-length code so that each frame may be expanded to a normal television signal in 33 milliseconds. Sequences of these images are used to generate real-time cartoons without the use of photography.

The success of this compaction scheme depends on the existence of coherence in the synthesized image. Normal television scenery contains much fine detail in the background to make this scheme successful. However, synthetic scenes, such as used in computer aided design of object assemblies, generally are preferred without such busy backgrounds. In these cases the encoding scheme presented here can be, and has been, used with great success. The particular hardware described here is designed to operate on the PDP-11 Unibus using a normal DMA controller which limits the usable data rate to 800,000 bytes per second. Thus each TV frame must be described with about 26,000 bytes of data.

Objects having several hundred facets, with or without smooth shading, and objects having curved surfaces, such as space-filling atomic models can be displayed. All these objects show image coherence such that from a visible image point a simple prediction can be made of the succeeding visible points. Special digital hardware operating at television display frequencies is used to generate groups of display points.

REFERENCES

A. Literature

(1) J. Staudhammer, "Computer Generation of Real-Time Colored Three-Dimensional Objects", Proc. 7th Hawaii Conf. on Inf. Sci., January 1974

(2) D.J. Ogden and J. Staudhammer, "Computer Graphics for 3-D Object Images", Computers and Graphics, Vol. 1, No. 1, June 1975

(3) J. Staudhammer, "Software for Real-Time Image Generation", Proc. COMPSAC-77, November 1977

(4) J.F. Eastman, "An Efficient Scan Conversion and Hidden Surface Removal Algorithm", Computers and Graphics, Vol. 1, No. 2/3, December 1975

(5) A.J. Myers, "A Digital Video Information Storage and Retrieval System", Computer Graphics, Vol. 10, No. 3, Summer 1976

(6) R.J. Hackathorn, "ANIMA-II: A 3-D Color Animation System", Computer Graphics, Vol. 11, No. 2, Summer 1977

(7) C. Csuri, "3-D Computer Animation", in Advances in Computers, Academic Press, 1977

(8) J.T. Whitted, "A Scan Line Algorithm for Computer Display of Curved Surfaces", Computer Grpahics, Vol. 13, No. 3; also: Ph.D. Dissertation, North Carolina State University, Raleigh, NC, August 1978

(9) J. Staudhammer, "Display of Multidimensional Objects (four and Higher Dimensions)", Computer Graphics, Vol. 9, No. 2, Summer 1975

(10) J. Staudhammer, "On Display of Space Filling Atomic Models in Real-Time", Computer Graphics, Vol. 13, No. 3, August 1978

B. Video Tapes

(T1) C. Csuri, "Demonstration of ANIMA", Computer Graphics Research Group, Ohio State University, Columbus, OH, Summer 1976

(T2) C. Csuri et al, "ANIMA-II", CGRG,OSU, Summer 1977

(T3) H. Moellering, "A Demonstration of the Real-Time Display of Three-Dimensional Cartographic Objects", Department of Geography, Ohio State University, Columbus, OH, Summer 1978

C. Picture Credits

Figures 1 - 5	H. Moellering, Ref. (T3)
Figure 6	C. Csuri
Figures 7,10	C. Csuri, Ref. (T1)
Figure 8,9	NCSU Signal Processing Laboratory
Figure 11	Ref. (9)
Figures 12,13	Ref. (10)

Note: Figures 1 - 5, 7, 10 are photographs of stop-motion single frames from re-recorded videotapes.

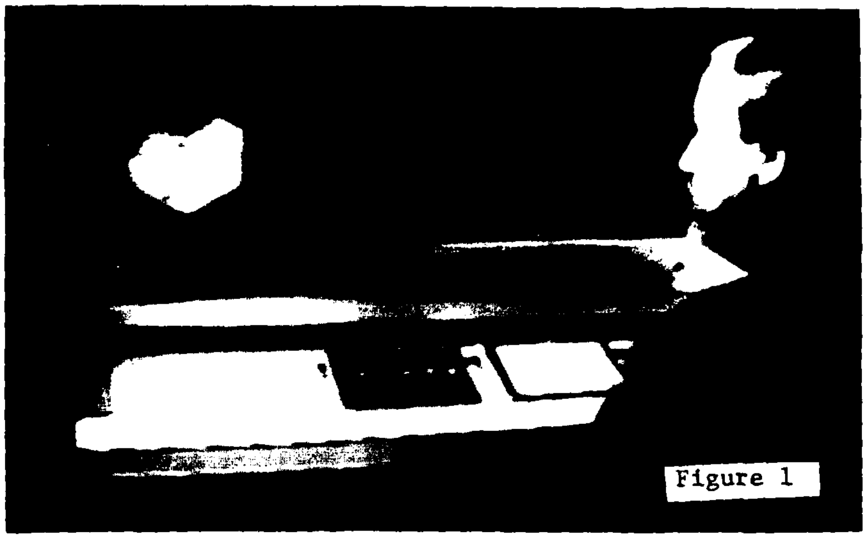


Figure 1

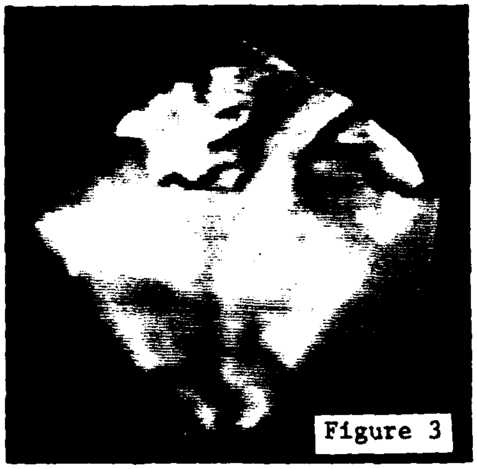


Figure 3

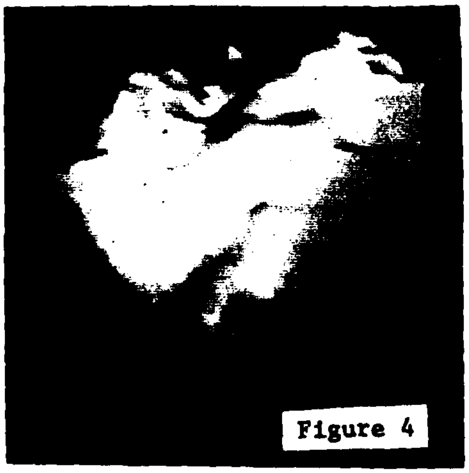


Figure 4

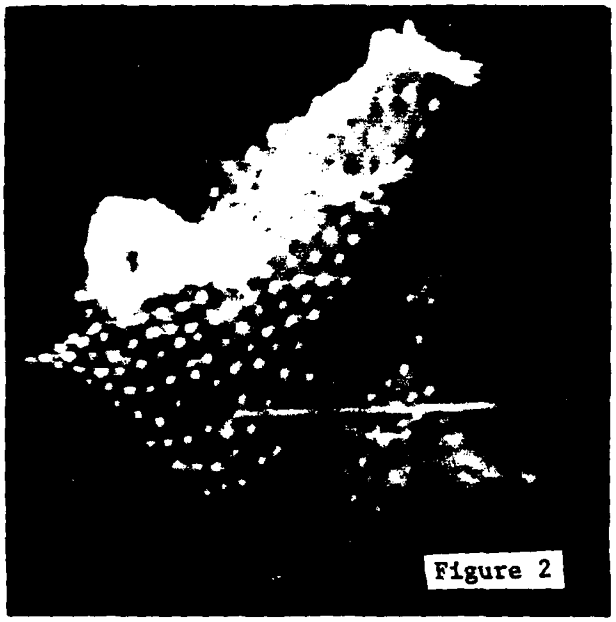


Figure 2

Fig. 1 View of Image Generation

Fig. 2 Wire Frame image Detail

Fig. 3 Scan Image from Fig. 1

Fig. 4 Rotated Object from Fig. 3

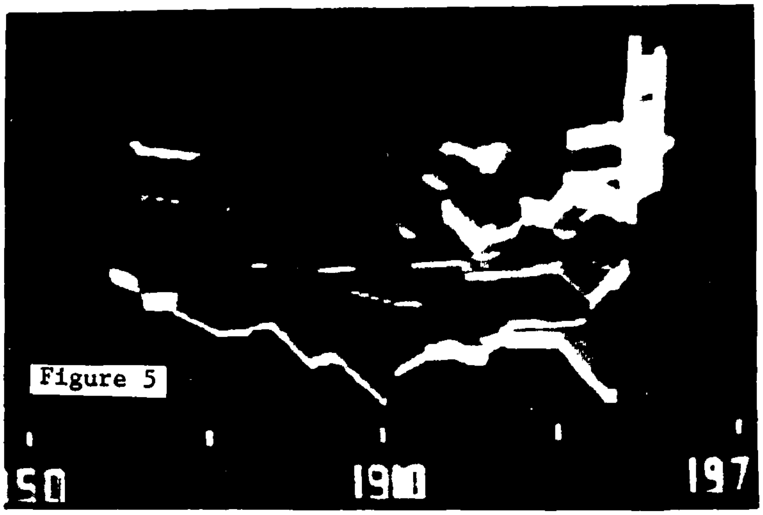


Figure 5

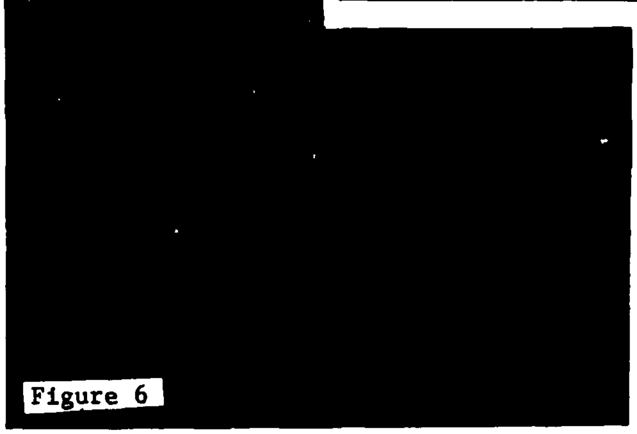


Figure 6

Fig. 5 Population Density Map

Fig. 6 Textured Surface

Fig. 7 Video Generator

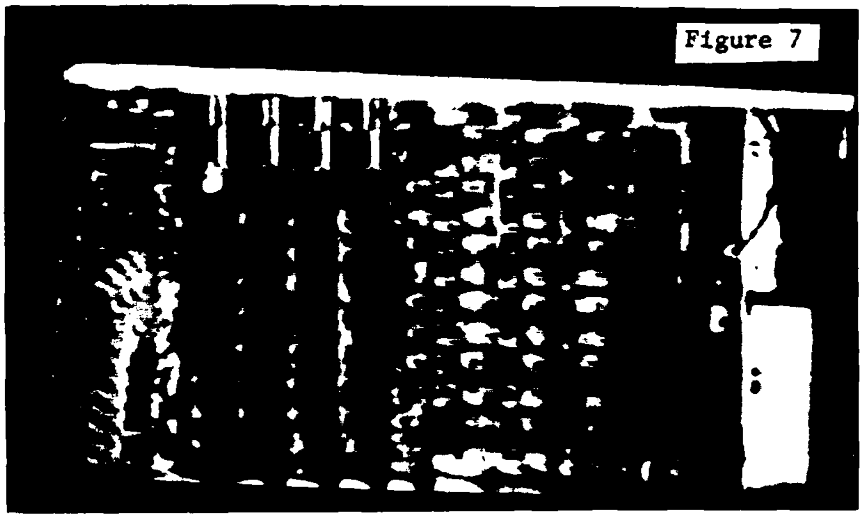


Figure 7

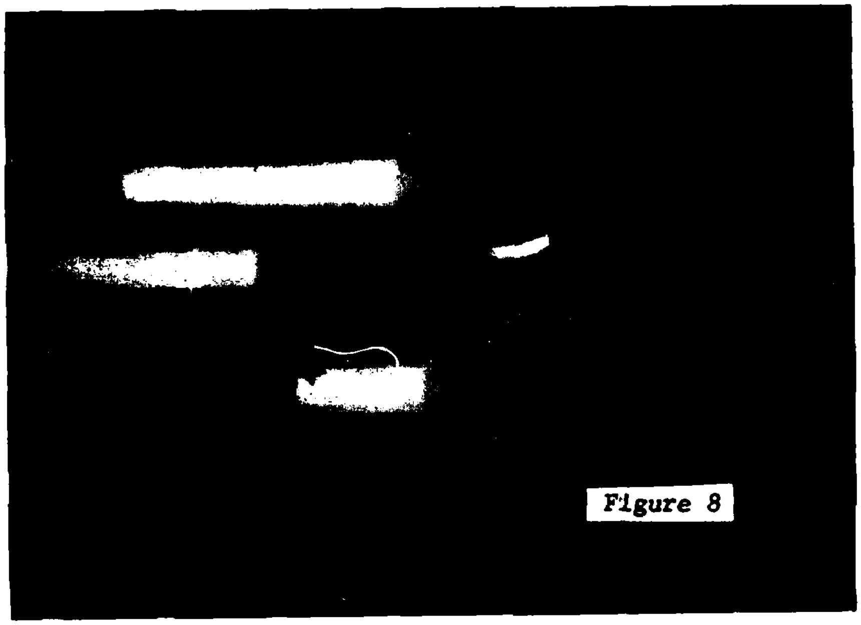


Figure 8

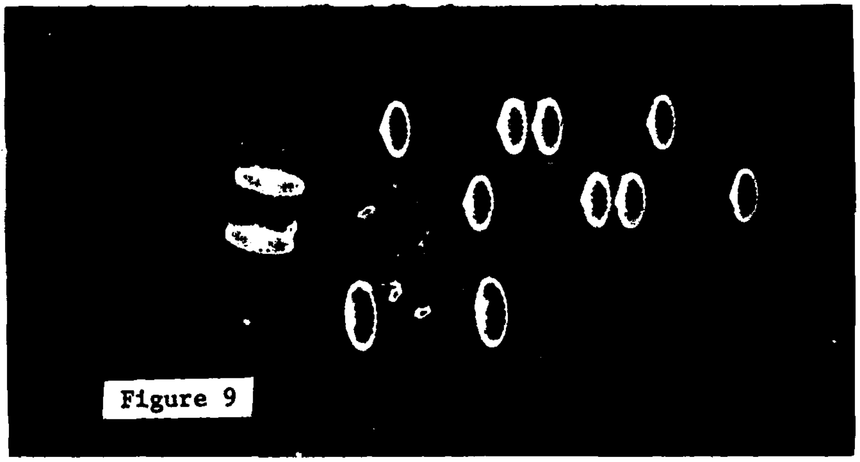


Figure 9

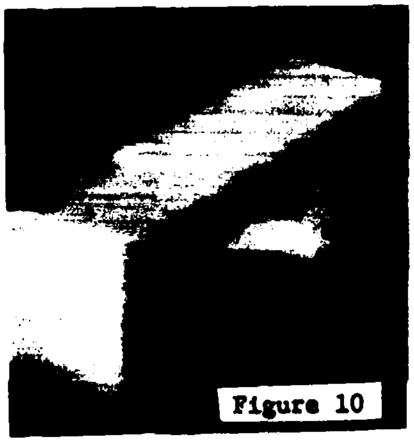


Figure 10

Fig. 8 Star Ship - Scan Image

Fig. 9 Star Ship - Wire Frame Image

Fig. 10 Detail of 3-D Object Image

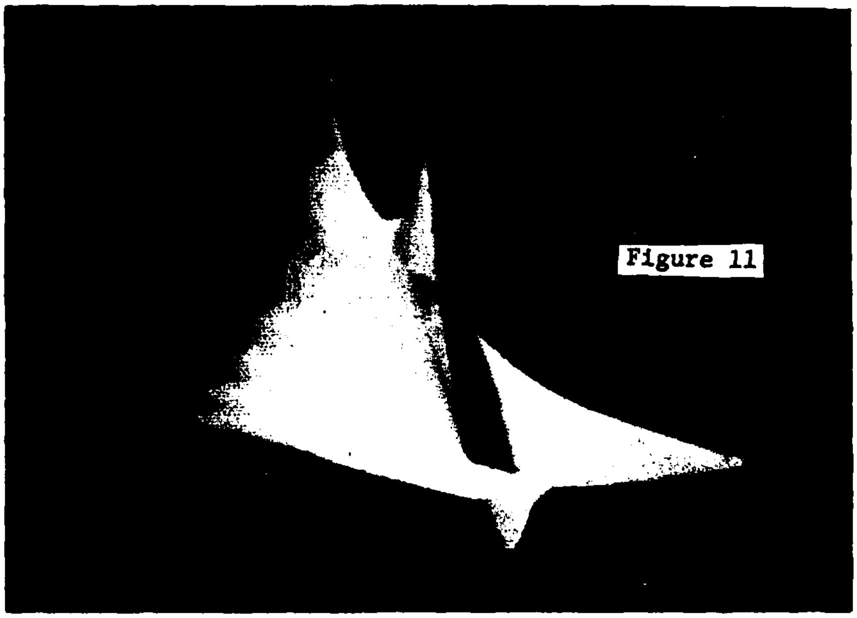


Figure 11

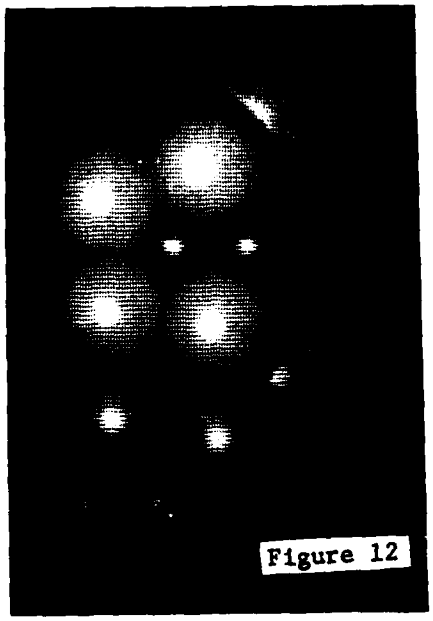


Figure 12

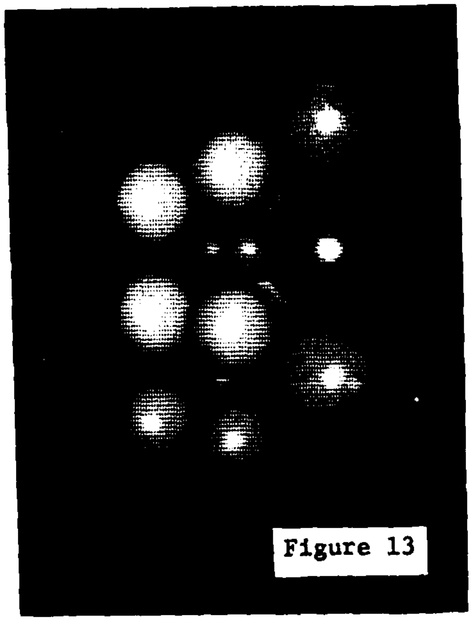


Figure 13

Fig. 11 Pole-Zero Function (Phase Angle in Color)

Fig. 12 Atomic Model - Spinel Molecule

Fig. 13 Atomic Radii Scaled from Fig. 12

TESTING THROUGH FORMALIZED METHODS of REQUIREMENTS & PROCEDURES

Norman J. Taupeka

CENTACS

TESTING THROUGH FORMALIZED METHODS OF REQUIREMENTS & PROCEDURES

SESSION CHAIRPERSON: Norman J. Taupeka

Chief, Systems Engineering Division
CENTACS

SESSION SUMMARY

This session examined the testing issue which is embodied throughout the entire life cycle process. Requirements analysis is the key for clearly stating the functional and operational aspects of a tactical data system. Structured techniques to address the functionality are described in the first paper, "A *Structure for Developing* Verifiable and Validatible Software Systems" by Jon C. Jervert, CENTACS. The elements of the materiel developer's needs (Project Manager or system developer) to provide for verification and validation, and hence reduction of risk and cost overrun and schedule extension were described in the second paper, "Role of a V&V Contractor in Development of Tactical Software Systems", by Lieutenant Colonel Charles R. Lindsey, Product Assurance and Test Directorate, CORADCOM. The specialized role of the independent tester in conjunction with the materiel developer to assess system performance and stress capabilities were addressed as a cohesive team effort to produce acceptable and fieldable tractical data systems. The third paper was "Software Testing at the System Level" by J. Gary Nelson, U.S. Army Test and Evaluation Command.

A Structure for Developing Verifiable and Validatable Software Systems

Jon C. Jervert

CENTACS

The purpose of this paper was to provide insight into the management, definition, specification and development of verifiable software systems.

Proceeding from a characterization of the management issues involved in software development a definition of verification and validation will be given. The role verification and validation plays in the software system life cycle was discussed. A definition for a verifiable software system structure was given and the software end-product was defined.

The relation between software life cycle milestones, guidelines and documentation for developing software specifications for systems within different phases of the DOD life cycle that will tend to produce a verifiable software end-product is given. Some contractual considerations concluded this paper.

1. Introduction:

The purpose of this paper is to provide insight into the management, definition, specification and development of verifiable software systems. Proceeding from a characterization of the management issues involved in software development the definition of verification and validation will be described. The role verification and validation plays in the life cycle will be given. A definition for a verifiable software system structure will be developed and the software end product will be defined.

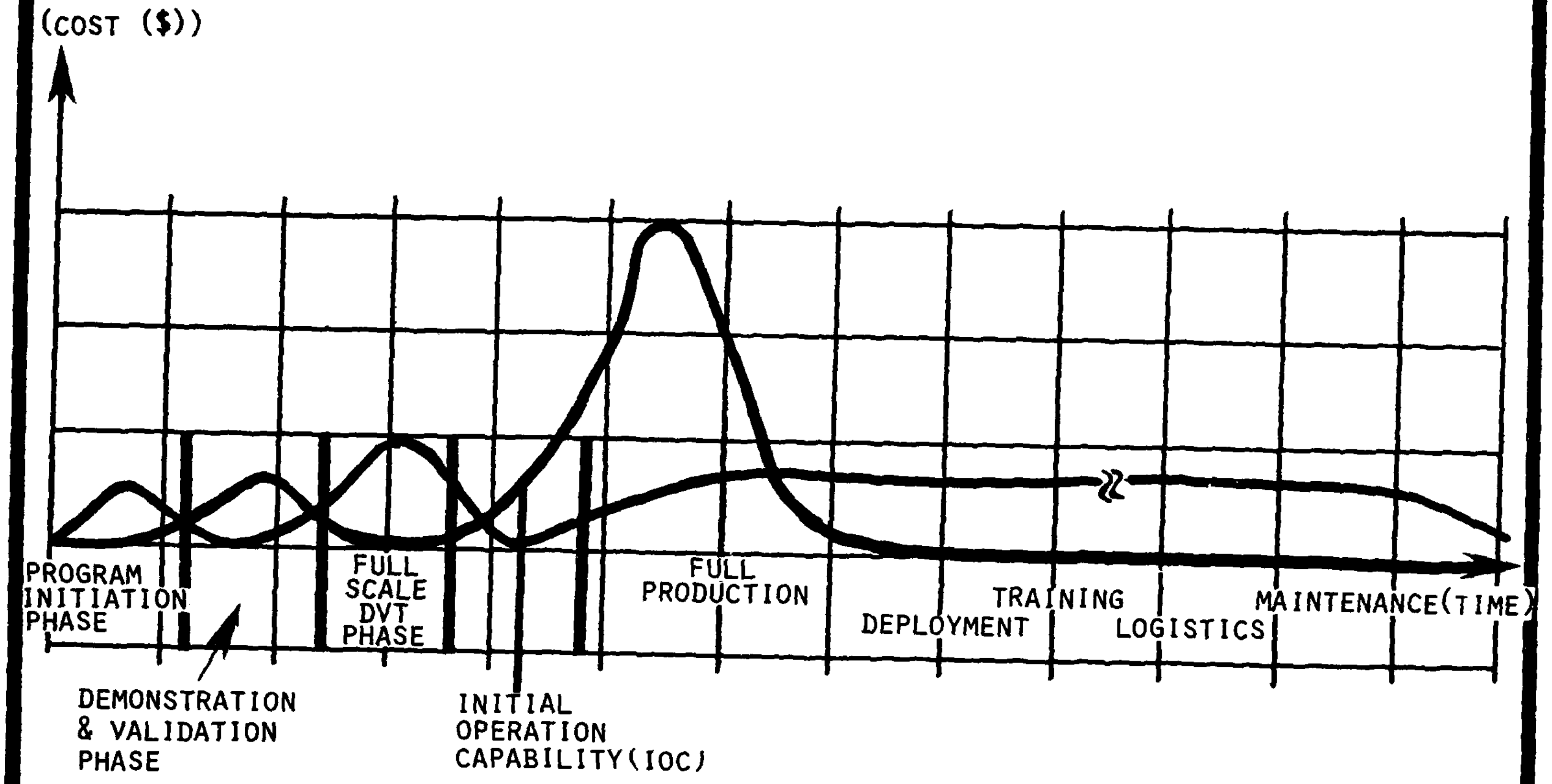
The relation between life cycle milestones, specifications and documentation and guidelines for developing software specifications for systems within the different phases of the DOD life cycle are given that will tend to produce a verifiable system. Contractual considerations will conclude this paper.

2. Relation Between the System Life Cycle and the Software Life Cycle:

2.1 Hardware vs Software Life Cycle: Before Verification and Validation can be discussed in relation to the structure and development of software systems the character of the problem we are attempting to solve must be defined in terms of the difference between hardware and software life cycle issues. The most productive way to demonstrate the difference between hardware and software is to characterize the shape of the resource utilization curve across the life cycle. Figure 2-1 depicts graphically the consumption of resources across the life cycle for a typical Army hardware system. Note that peak resource utilization occurs well after Initial Operational Capability (IOC). In typical software life cycles which have been characterized elsewhere (see Figure 2-2), the peak occurs substantially prior to IOC. When software consumes a major portion of the system life-cycle resources, it is necessary to get top management understanding and approval of early peak resource utilization. If this is not done, and the same IOC is attempted to be met, that IOC will be premature. This results in a fictitious operational capability wherein the user is led to believe that a real operational capability exists, when in fact it does not. Unless specifically planned for software development while under live operation will be traumatic.

In addition to the resource consumption function being different from that of hardware, the software life cycle has a different set of natural intrinsic milestones. Unless these are mapped properly into the total system life cycle, and made clearly visible, then the software part of the system can not be properly controlled.

THE ARMY LIFE CYCLE



HARDWARE LIFE CYCLE CONSUMPTION FUNCTION

CENTACS

Finally, every time software maintenance is performed the product specification is changed. Thus, software maintenance requires a much higher lever of management than its hardware counterpart.

These characteristics demonstrate that the mission of the Army System Developer is not only to produce a quality system that performs its function but also to provide for optimization of the systems resource consumption throughout a systems life cycle. One general tool that will help us meet our mission is broadly termed verification and validation.

For the purposes of this paper verification will be defined to mean the procedures and criteria necessary to insure consistant progress between and within development phases of the Army Life Cycle (FIG 2-1) and is the main subject of this paper. Validation will be defined as the test and evaluation necessary to insure the performance of the systems function from both the users and developers point of view.

2.2 Measuring Program Success: Having defined the problem, we now consider how the project manager can achieve practicl solutions. Questions to be answered are: What are the controls that shape the curves? How can we predict their responses? And, how can we validate our predictions in a changing environment? These questions have been approached by comparing software vendors in a competitive environment (see Figure 2-3), and some general conclusions have been drawn.

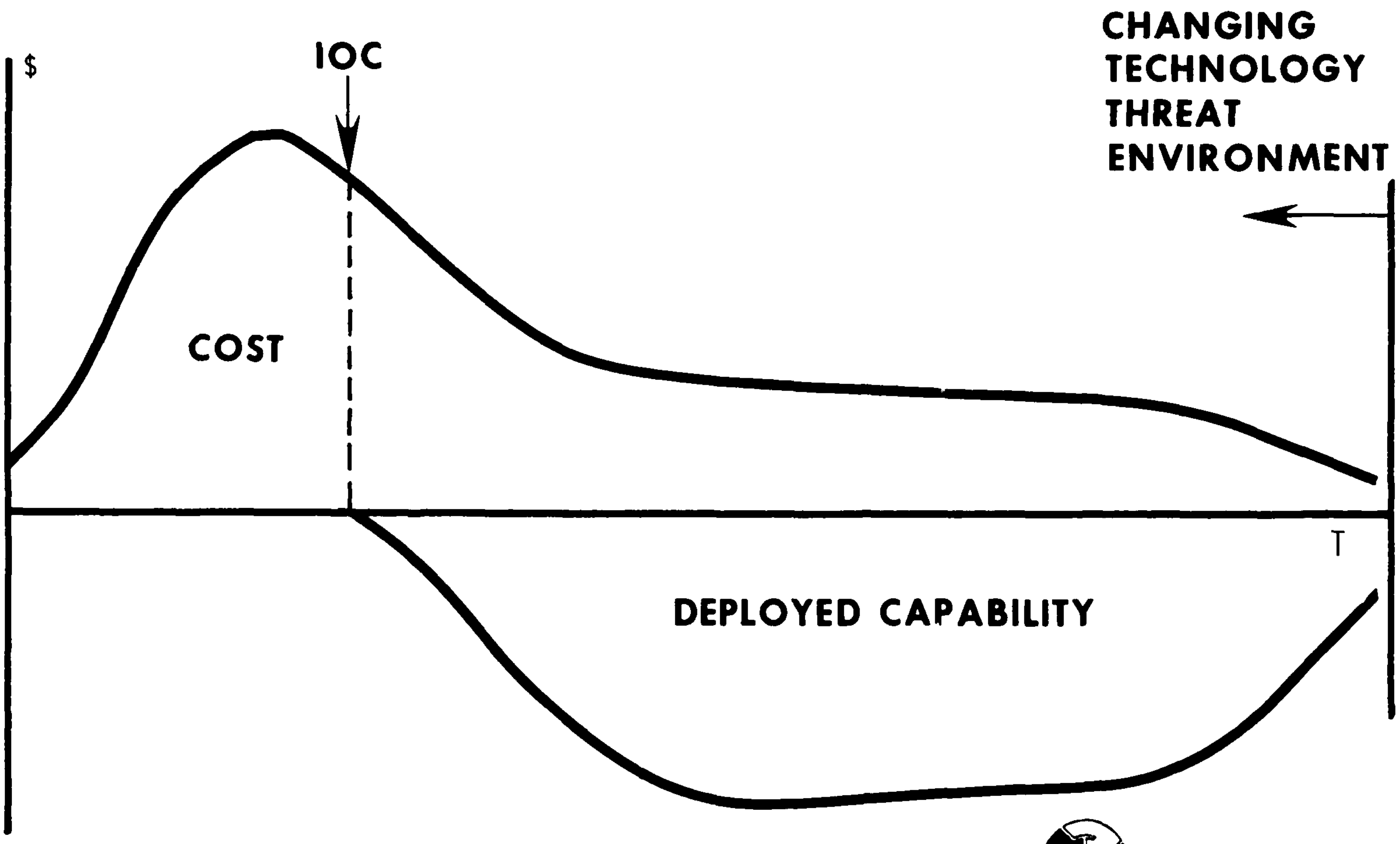
- Certain software developers have been consistently more successful than others.

- There is a common view that risk can be reduced by going slowly, particularly if management is inexperienced. However, going slowly may not minimize risk in the way defined above and, in fact, can decrease return-on-investment (ROI) dramatically.

- The steep rate of resource utilization by developer Number 1 requires a knowledge of how to effectively use those resources -- in other words, management experience and in applying tools like verification and validation.

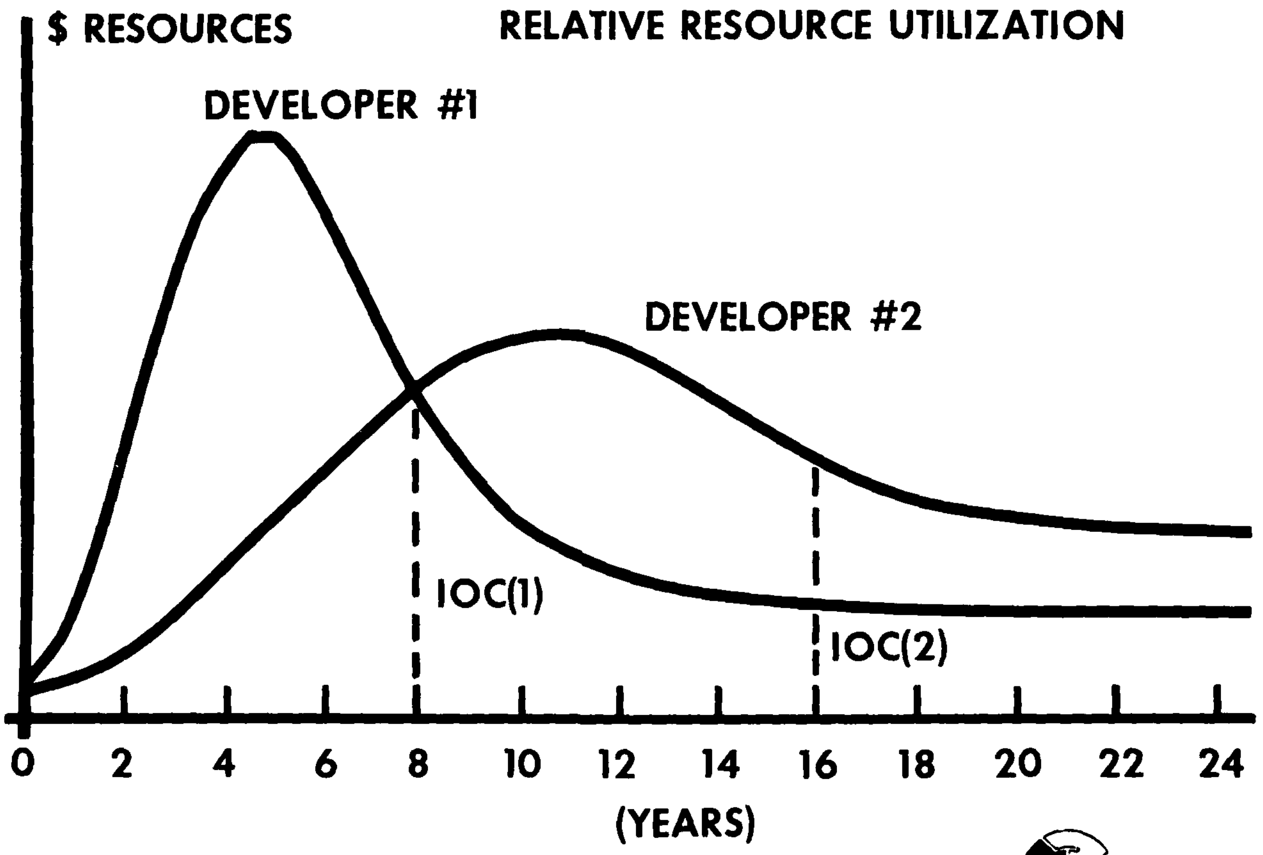
- It is important to understand how and where software fits into the life cycle of total systems and to what degree software life-cycle management influences total system management.

TYPICAL SOFTWARE LIFE CYCLE



CENTACS

COMPETATIVE DEVELOPER CHARACTERISTICS



CENTACS

2.3 Software Life Cycle Milestones: Economic application of Verification and Validation techniques and project success depend on life cycle planning and precise system documentation.

Successful life-cycle planning must provide for an incremental commitment of resources. Increment size can be determined by the time frame and resources required to achieve a measurable milestone, and the risk of not achieving that milestone. A set of milestones which have been used successfully in the past by various software developers are offered below. These milestones segment that software development cycle into sequential phases. Each phase should be terminated with thorough documentation of work completed, a critical review, a detailed plan for the next phase, and an updated overall plan for the remainder of the project. A go-no decision can then be made regarding commitment of resources for the next phase. If a go decision is made, changes resulting from review are formally incorporated into a continuation plan and the next phase initiated. Figure 2-4 shows how these software milestones map into the four DOD life cycle phases.

The following definitons of management milestones for controlling software life cycles are offered.

1. Project Definition - The problem to be solved must be defined in general terms. Basic user objectives and constraints must be agreed upon along with an overall plan and gross estimate of resources required for their satisfaction.

2. Functional Analysis and Specification - User functional requirements must be analyzed and detailed functional specifications developed for the system.

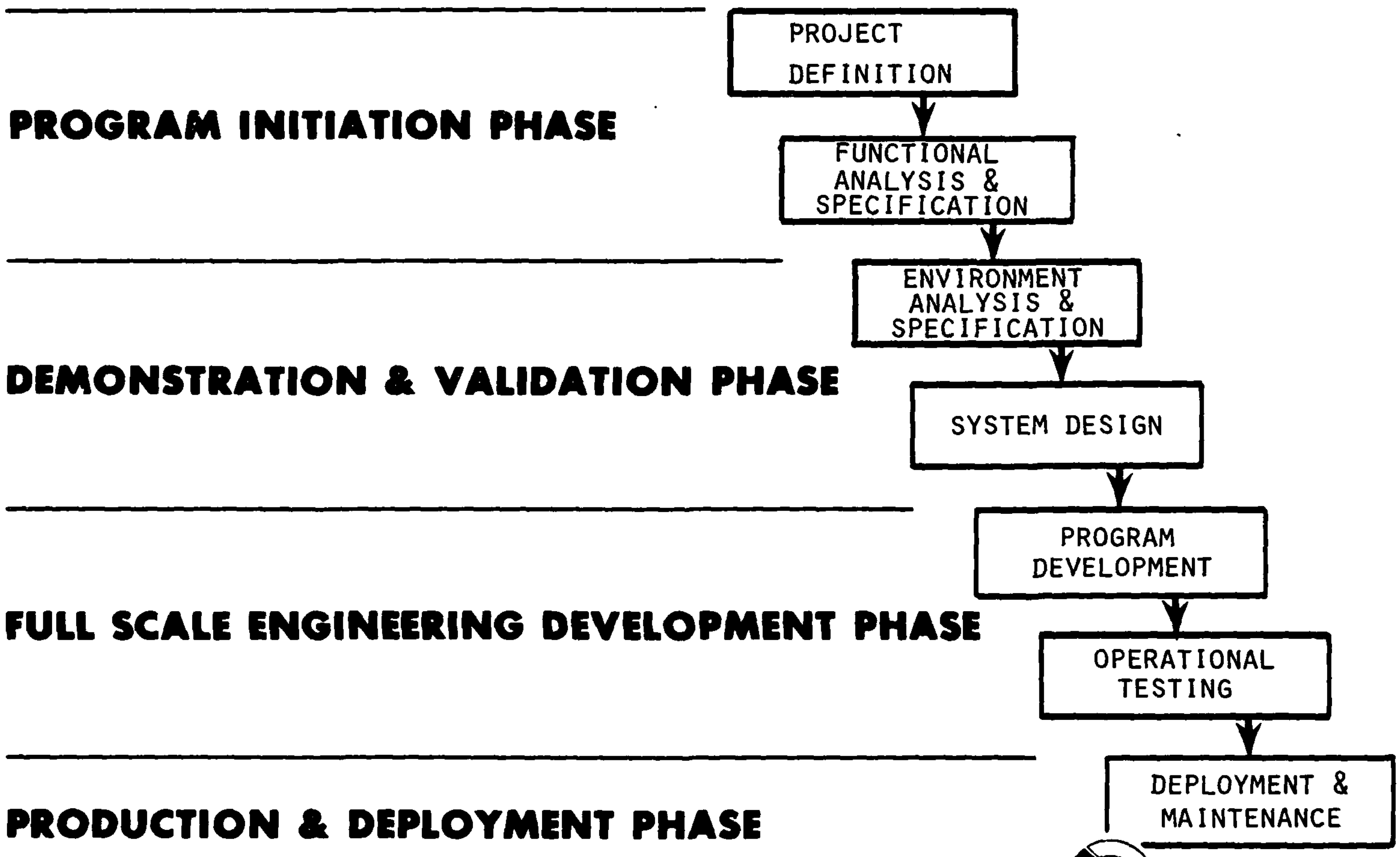
3. Environment Analysis and Specification - Implementation and development requirements, such as equipment configuration, languages, and support software to be used for both the development and operational environments, must be produced.

4. System Design - Detailed sytem design must be completed to the module level. A detailed documentation set and a detailed development and testing schedule must also be completed.

5. Program Development - All programs must be coded and integrated into a working system. A regression test set must be completed and available for quality control of future modifications. A training package must also be completed.

6. Operational Testing - Initial users utilize the complete system in a carefully maintained operational environment. Quality control procedures for supporting the maintenance environment are fully implemented.

SOFTWARE
DEVELOPMENT LIFE CYCLE



PROGRAM INITIATION PHASE

DEMONSTRATION & VALIDATION PHASE

FULL SCALE ENGINEERING DEVELOPMENT PHASE

PRODUCTION & DEPLOYMENT PHASE

CENTACS

7. Deployment and Maintenance - The system is operationally deployed and maintained for multiple live installations which may be geographically separated. The training package is used for bringing up live installations. The sytem is corrected, refined, and enhanced until obsolescence.

In the introduction, it was indicated that software is different from hardware. It turns out that the set of hardware system life-cycle milestones, activities, and events within the Army do not require much change in terms of names and general achievement in order to be used for software. What they do require are proper interpretation and a different set of specific measurement criteria for their attainment. This requirement for management milestones and standards peculiar to software is very important to recognize.

Verification and Validation tools cannot be utilized economically without application of good planning, strong milestone management and adhearance to precise documentation.

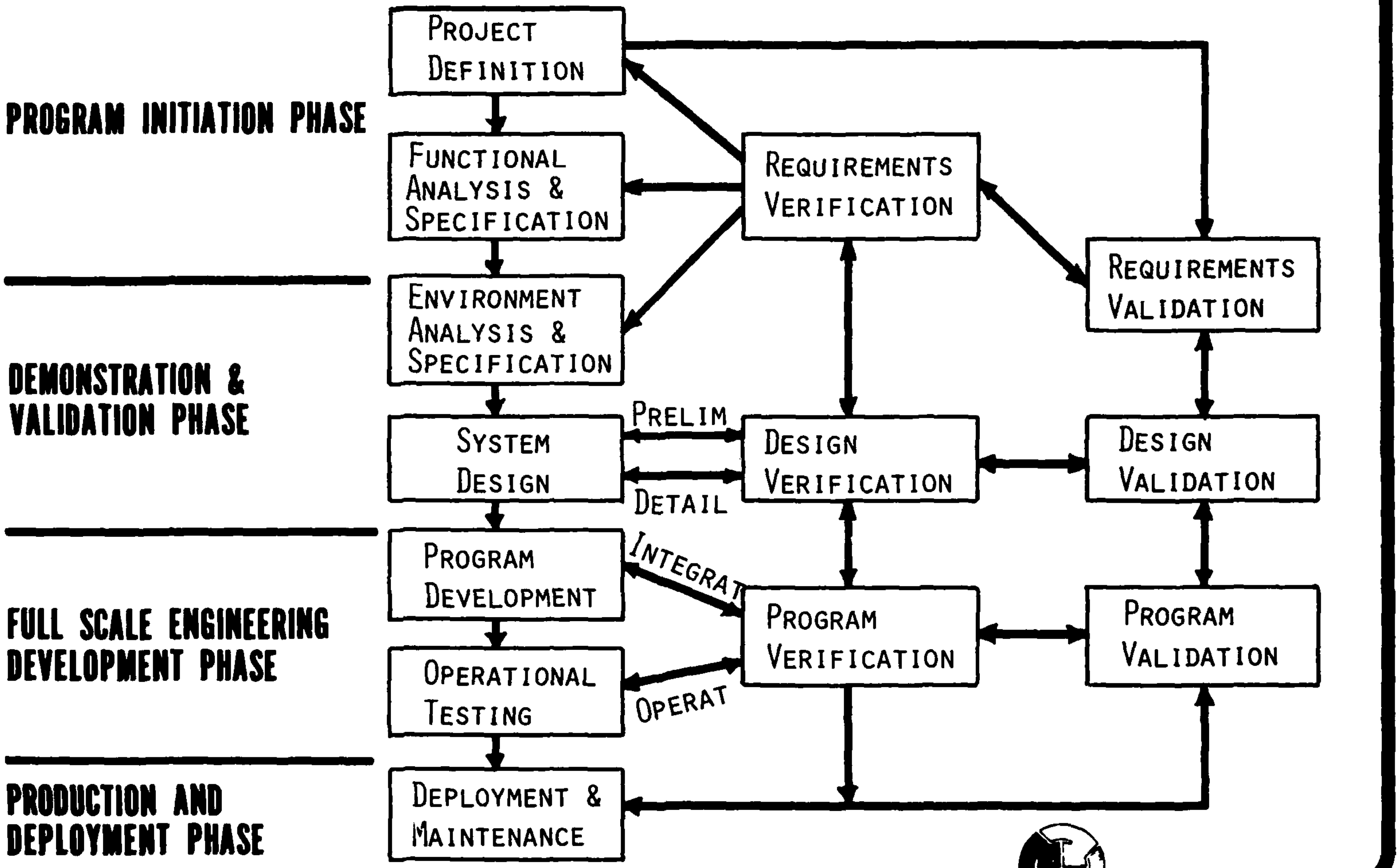
2.4 Verification and Validation as a Yardstick: Verification and Validation provides a set of tools that allow a program manager to judge the quality of program progress during the life cycle. Figure 2.5 indicates the major Verification and Validation milestones in relation to the major development life cycle milestones. The verification of systems requirements, design requirements and program requirements and the validation of these requirements are carried out by completing the general tasks of figure 2-6. These tasks can be successfully carried out with or without the utilization of automated tools. However significant economics of resource can be achieved by the application of the general tools outlined in figure 2-7.

The effectiveness and cost of any Verification and Validation effort is only as good as the planning that integrates it into a program and a software sytem structure that allows for a documented software end product. Without sufficient planning, early in the program, and documentation maintained throughout the program, the result could be minimal and the program cost doubled. Verification and Validation will not replace good management or good system engineering practices it only provides a tool to increase efficiency and productivity.

3. Verifiable Software Systems:

Our purpose is to develop a concept for specifying complete software systems in such a manner that the verification and validation tools outlined in section 2.4 can be applied to reduce life cycle costs, as well as insure proper system function.

V&V IN THE DEVELOPMENT LIFE CYCLE



PROGRAM INITIATION PHASE

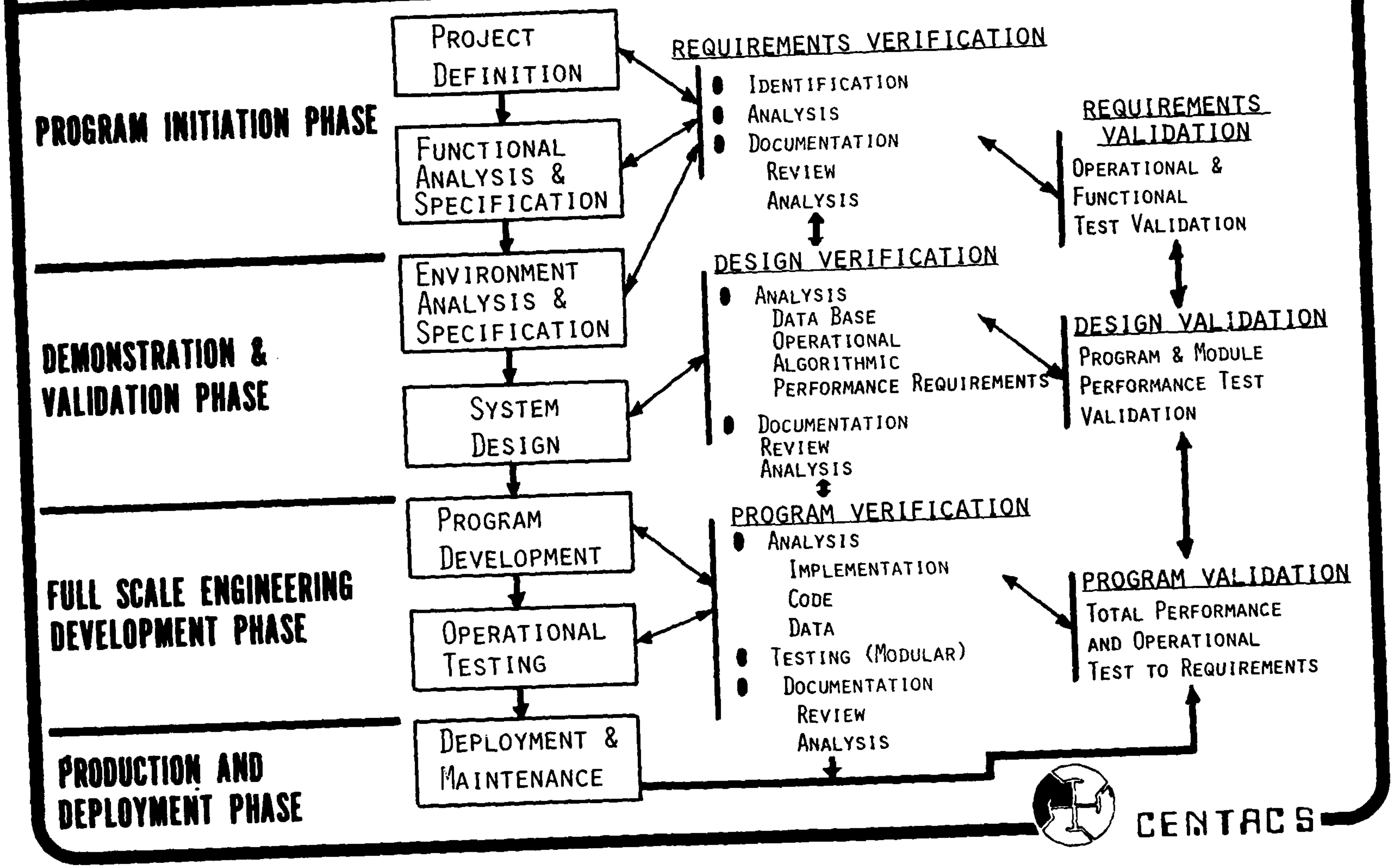
DEMONSTRATION & VALIDATION PHASE

FULL SCALE ENGINEERING DEVELOPMENT PHASE

PRODUCTION AND DEPLOYMENT PHASE

CENTACS

V&V TASKS



PROGRAM INITIATION PHASE

DEMONSTRATION & VALIDATION PHASE

FULL SCALE ENGINEERING DEVELOPMENT PHASE

PRODUCTION AND DEPLOYMENT PHASE

PROJECT DEFINITION

FUNCTIONAL ANALYSIS & SPECIFICATION

ENVIRONMENT ANALYSIS & SPECIFICATION

SYSTEM DESIGN

PROGRAM DEVELOPMENT

OPERATIONAL TESTING

DEPLOYMENT & MAINTENANCE

REQUIREMENTS VERIFICATION

- IDENTIFICATION
- ANALYSIS
- DOCUMENTATION
  REVIEW
  ANALYSIS

DESIGN VERIFICATION

- ANALYSIS
  DATA BASE
  OPERATIONAL
  ALGORITHMIC
  PERFORMANCE REQUIREMENTS
- DOCUMENTATION
  REVIEW
  ANALYSIS

PROGRAM VERIFICATION

- ANALYSIS
  IMPLEMENTATION
  CODE
  DATA
- TESTING (MODULAR)
- DOCUMENTATION
  REVIEW
  ANALYSIS

REQUIREMENTS VALIDATION

OPERATIONAL & FUNCTIONAL TEST VALIDATION

DESIGN VALIDATION

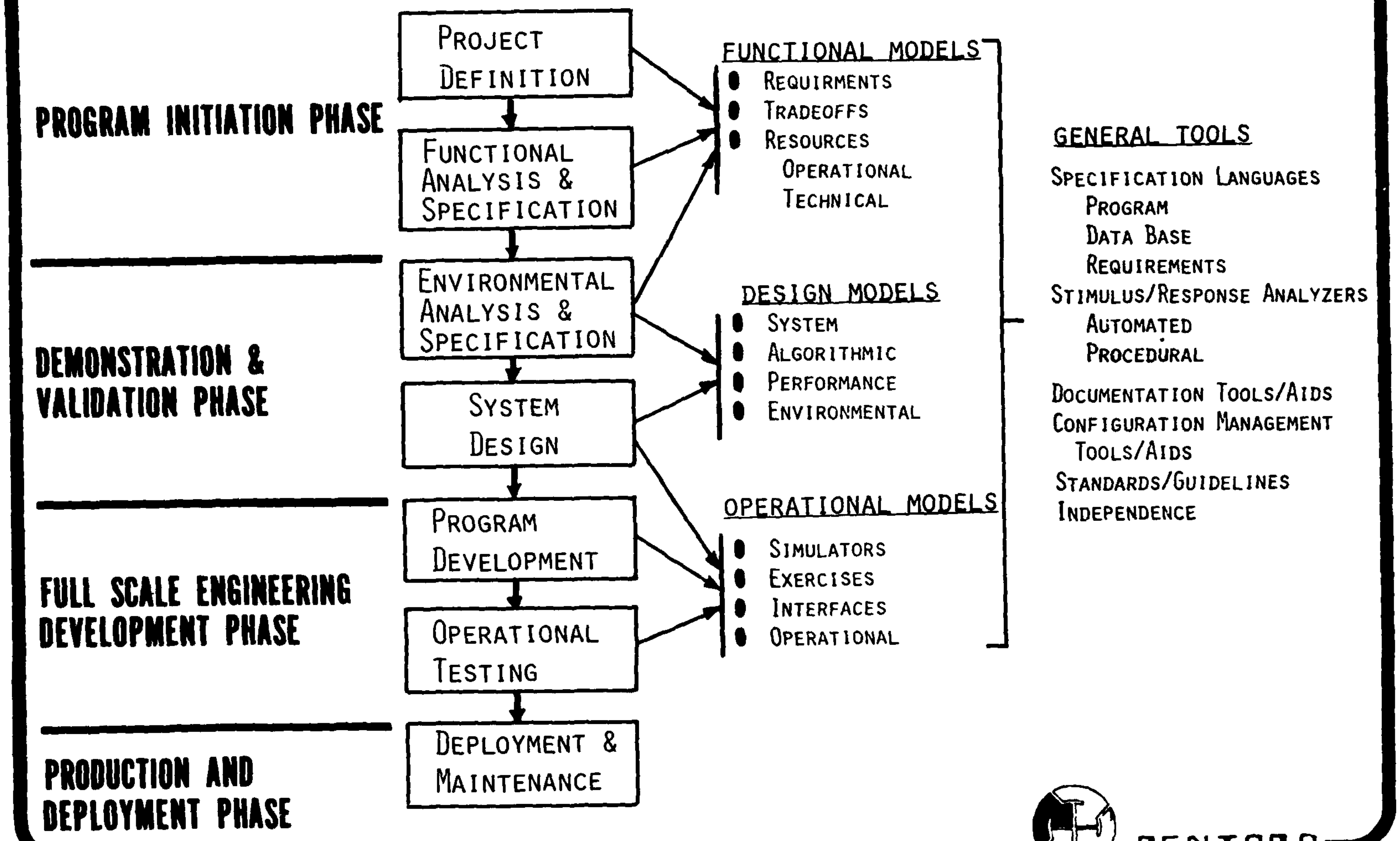
PROGRAM & MODULE PERFORMANCE TEST VALIDATION

PROGRAM VALIDATION

TOTAL PERFORMANCE AND OPERATIONAL TEST TO REQUIREMENTS

CENTACS

V&V TOOLS



PROGRAM INITIATION PHASE

DEMONSTRATION & VALIDATION PHASE

FULL SCALE ENGINEERING DEVELOPMENT PHASE

PRODUCTION AND DEPLOYMENT PHASE

FUNCTIONAL MODELS
- REQUIRMENTS
- TRADEOFFS
- RESOURCES
  - OPERATIONAL
  - TECHNICAL

DESIGN MODELS
- SYSTEM
- ALGORITHMIC
- PERFORMANCE
- ENVIRONMENTAL

OPERATIONAL MODELS
- SIMULATORS
- EXERCISES
- INTERFACES
- OPERATIONAL

GENERAL TOOLS

SPECIFICATION LANGUAGES
  PROGRAM
  DATA BASE
  REQUIREMENTS
STIMULUS/RESPONSE ANALYZERS
  AUTOMATED
  PROCEDURAL
DOCUMENTATION TOOLS/AIDS
CONFIGURATION MANAGEMENT
  TOOLS/AIDS
STANDARDS/GUIDELINES
INDEPENDENCE

CENTACS

3.1 Verifiable Software Systems Structure: It is our intent to develop, based on the definition of figure 3-1, a structure to specify software systems which are major components of a complete defense systems. Such a software system must meet all of the complete system's operational requirements and must be maintainable. To develop and maintain a software system it is necessary to decompose it inot subsystems as shown in Figure 3-2. There are two major parts of the software system, host and target. The host software system provides the means to develop and maintain the target software system which will operate in the field. Each system may be resident on different machines.

3.2 Host Software: The subsystems within the host system shown in Figure 3-2 are defined as follows:

Host System Software (A). It contains:

* Operating System
* System Utilities
* Programming Systems (dompilers, assemblers, programming, and debugging aids, etc.)
* Library Facilities

Host Training Software (B). It is composed of software for training which cannot be accomplished in the field. It includes software for both maintenance and operation training.

Host Hardware Diagnostic Software (C). This subsystem consists of the software needed for host hardware diagnostics and maintenance. It does not include any software diagnostics, but can include firmware diagnostics.

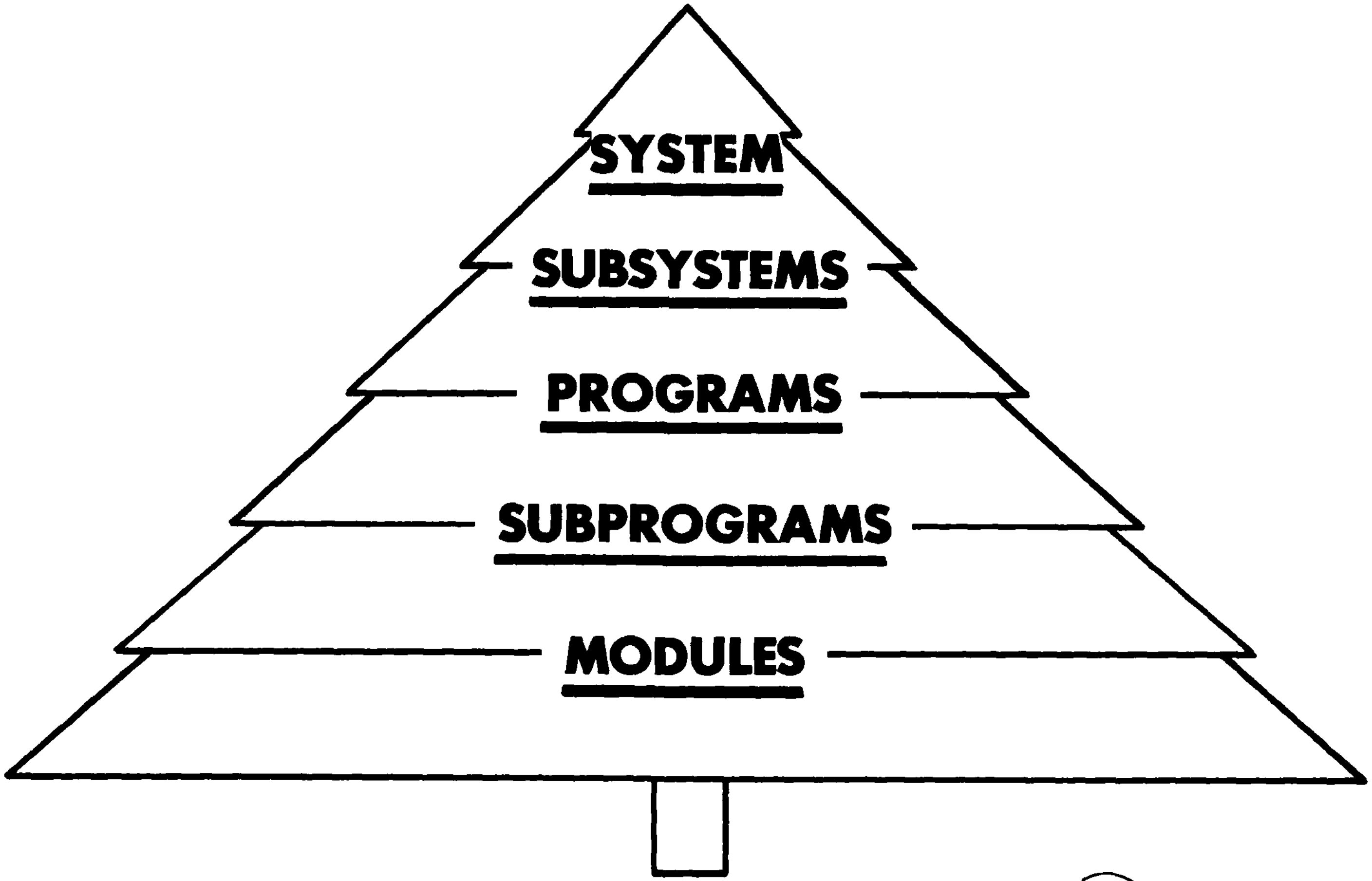
3.3 Target Software: The target software subsystems should all be operational on the host system, with the exception of the hardware diagnostics. The target software subsystems (Figure 3-2) are defined as the following:

Executive Software (D). This is the fielded executive (operating system) that provides a software operational environment.

Application Software (E). This subsystem contains the software which provides the functional capability of the operational target system such as:

- intelligence
- fire control
- operations
- etc.

THE SOFTWARE SYSTEM TREE



SYSTEM

SUBSYSTEMS

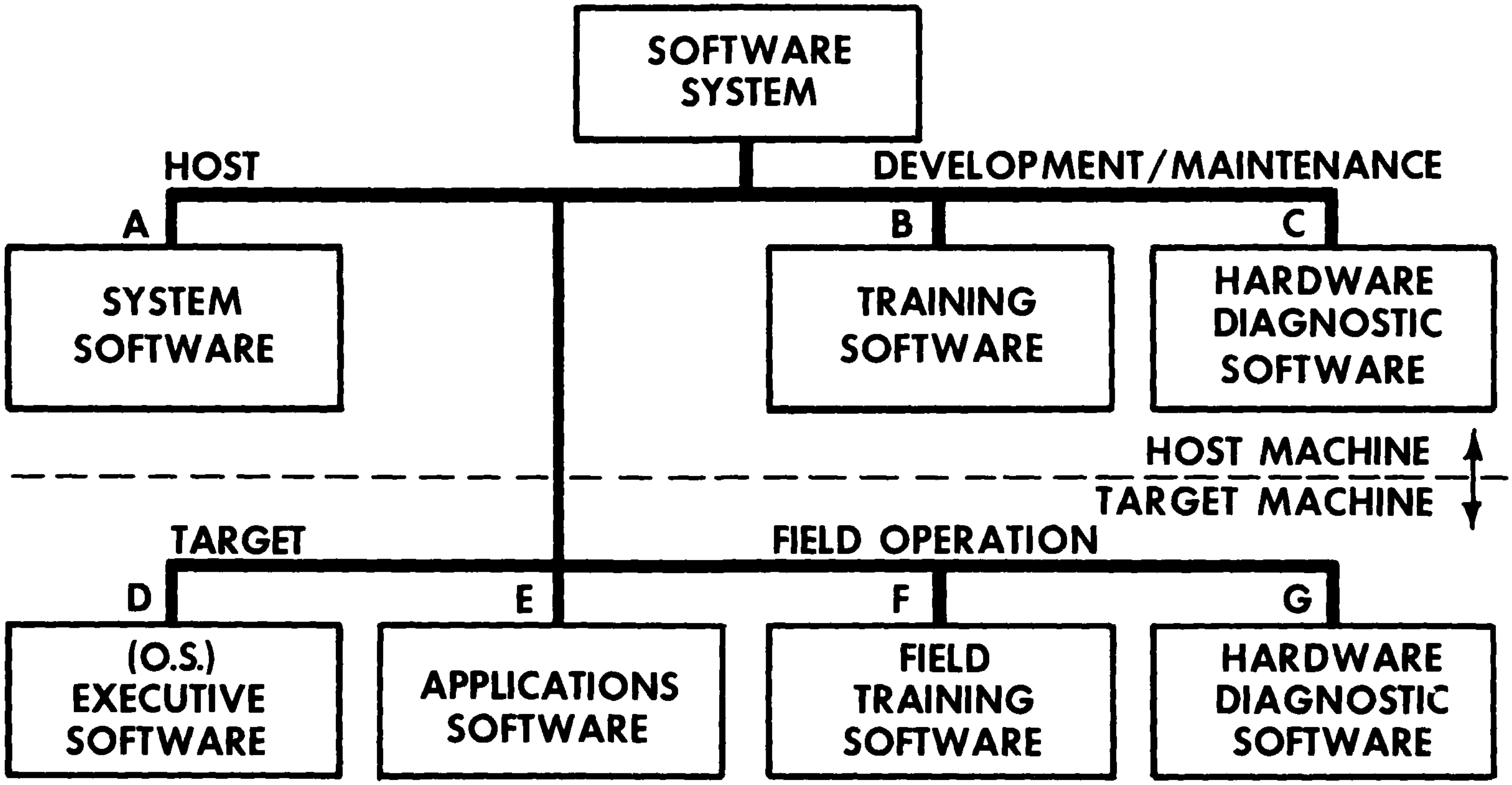
PROGRAMS

SUBPROGRAMS

MODULES

CENTACS

SOFTWARE SUBSYSTEMS



CENTACS

There may be several application software subsystems within a system (i.e., the G1, G2, G3, G4, etc.).

Field Training Software (F). This subsystem includes the software to provide user training in the field, limited by what the target system can accommodate.

Hardware Diagnostic Software (G). This contains the field software to diagnose target hardware failures. Software tests sets are not included here.

4. Definition of the Software End Product:

A verifiable software system must include a set of well structured elements which are completely documented in detail. Each subsystem described in paragraph 3.2 is decomposed into a software end product, as shown in Figure 4-1.

The software end product consists of three different elements.

* External Documentation Library

* Program Library

* Test Library

These elements are defined below.

4.1 External Documentation Library: This library is divided into two document types, reference Figure 4-2.

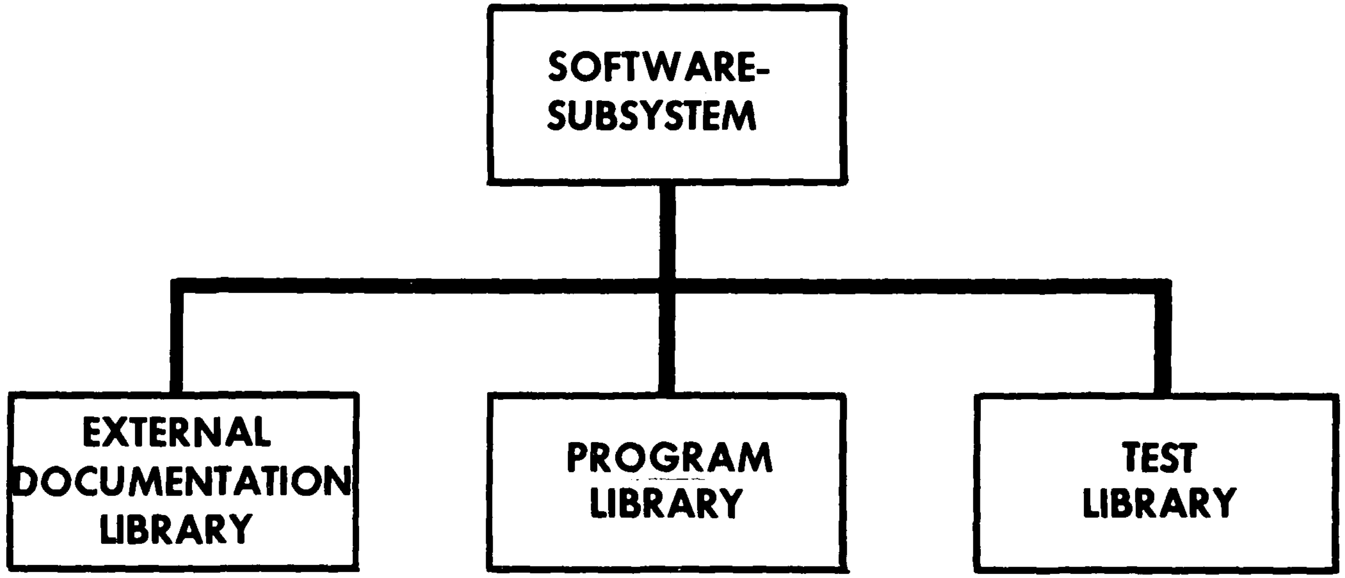
- System Outline

- Operational Procedures Documents

- Software Maintenance Documents

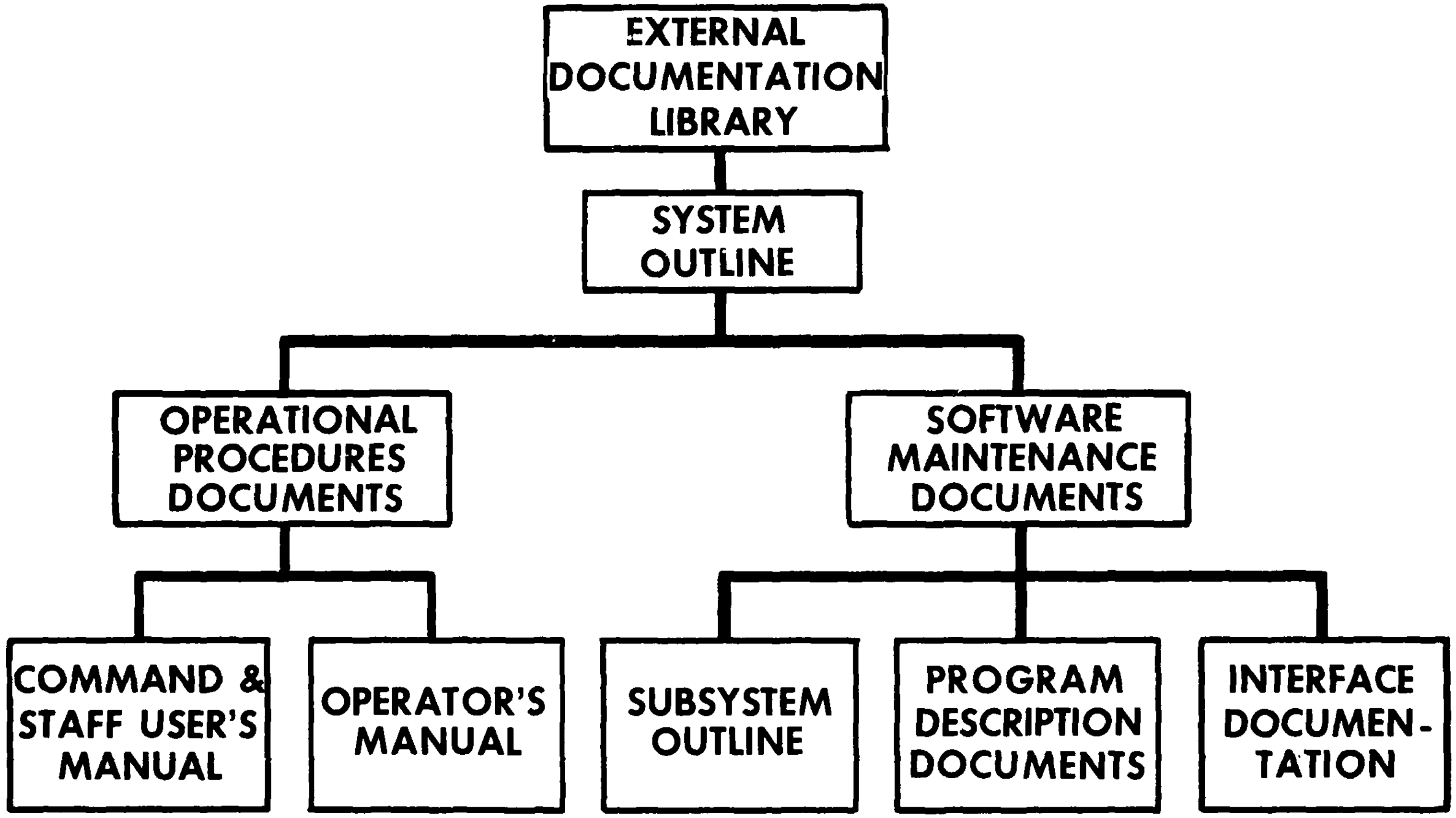
The System Outline gives a total software overview. The main purpose of the Operational Procedures Documents are to give an overview of each subsystem's capabilities, general guidance for the user and provide the basis for discussing problems between user and maintainer. The Software Maintenance Documents are the tools necessary for software technicians to maintain each subsystem.

THE SOFTWARE END PRODUCT



CENTACS

EXTERNAL LIBRARY



EXTERNAL DOCUMENTATION LIBRARY

SYSTEM OUTLINE

OPERATIONAL PROCEDURES DOCUMENTS

SOFTWARE MAINTENANCE DOCUMENTS

COMMAND & STAFF USER'S MANUAL

OPERATOR'S MANUAL

SUBSYSTEM OUTLINE

PROGRAM DESCRIPTION DOCUMENTS

INTERFACE DOCUMENTATION

CENTACS

4.1.1 System Outline: The System Outline provides an overview of the total software system. It contains a block diagram and descriptive narrative for each software subsystem. A copy of the system outline will be contained in the External Documentation Library for each software subsystem.

4.1.2 Operational Procedures Documents: These documents consist of the following manuals:

- Command and Staff User's Manual. This manual provides the commander and his staff with a detailed description of how to use the system to fulfill their mission.

- Operator's Manual. This manual provides detailed guidance for operating the hardware (console etc.).

4.1.3 Software Maintenance Documents: There are three document types within this set.

- Subsystem Outline. Includes a subsystem overview with block diagrams and descriptive narratives.

- Interface Documentation. Describes and defines the Input/Output (I/O) formats, communication formats and data base formats. Refer to Figure 4-3.

- Program Description Documents. These documents describe (by flow charts and narratives) programs/modules and their interfaces within the subsystem.

4.2 Program Library. The Program Library consists of two parts, reference Figure 4-4:

- Coded Media, (sources code, object code)

- Listings, (containing the program statements and the internal program documentation)

4.3 Test Library. The Test Library consists of three major parts, reference Figure 4-5:

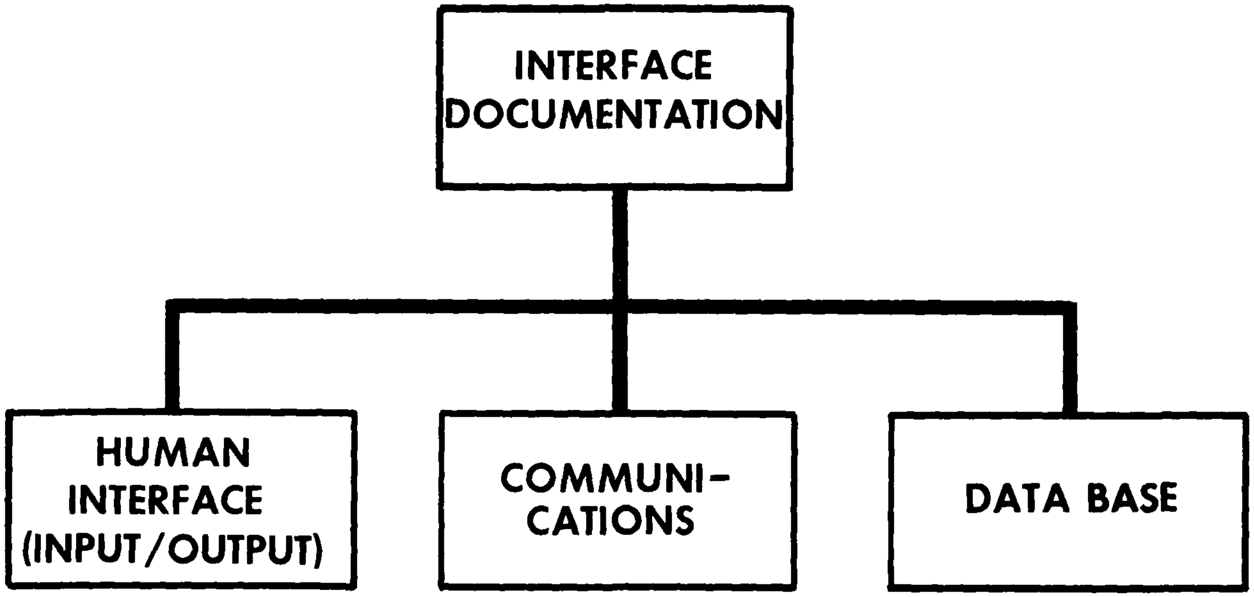
- Test Data Set

- Test Program

- Test Documentation (procedures and results)

The test library will provide the means to validate the systems function.

INTERFACE DOCUMENTATION



INTERFACE DOCUMENTATION

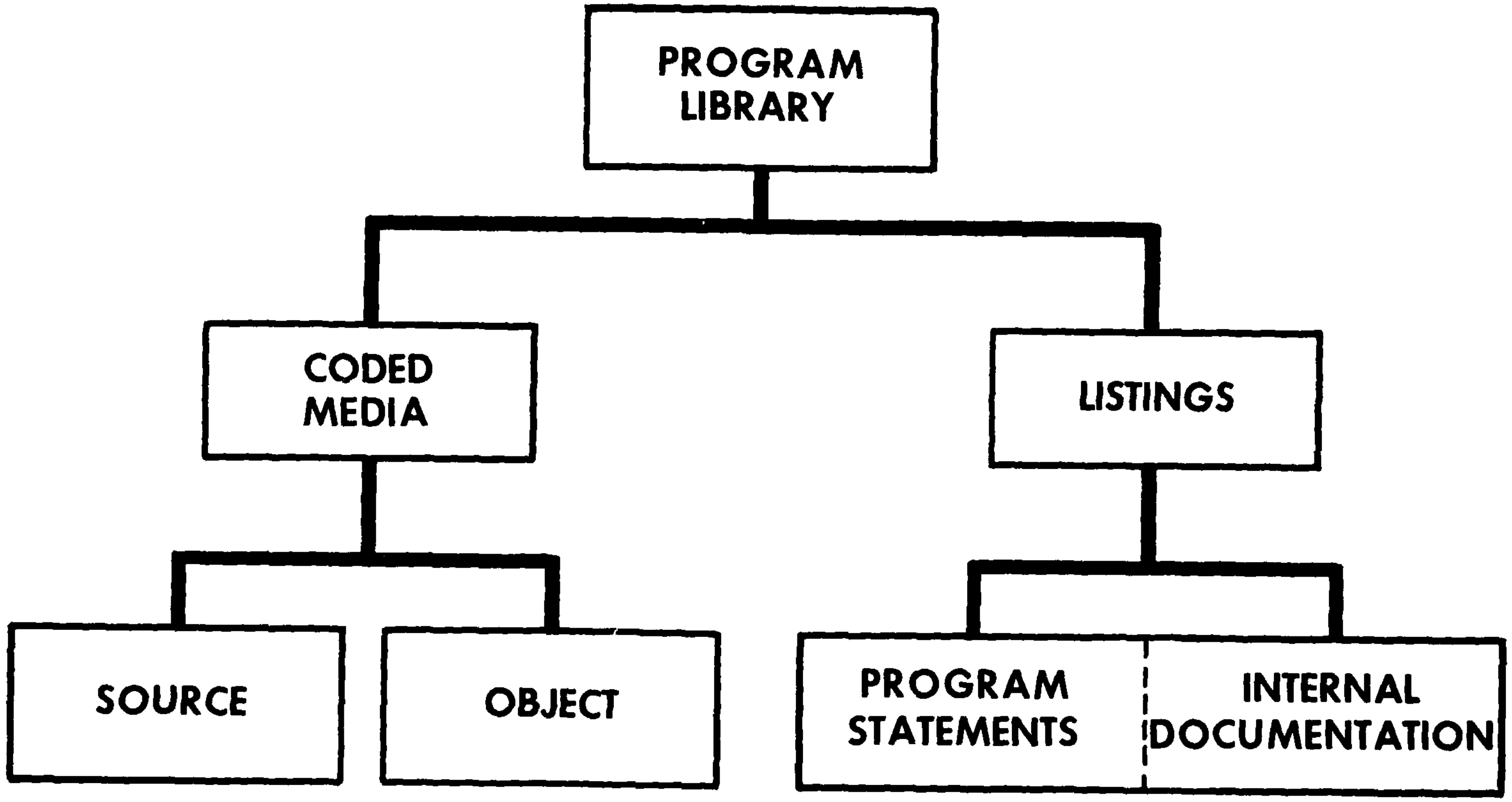
HUMAN INTERFACE (INPUT/OUTPUT)

COMMUNICATIONS

DATA BASE

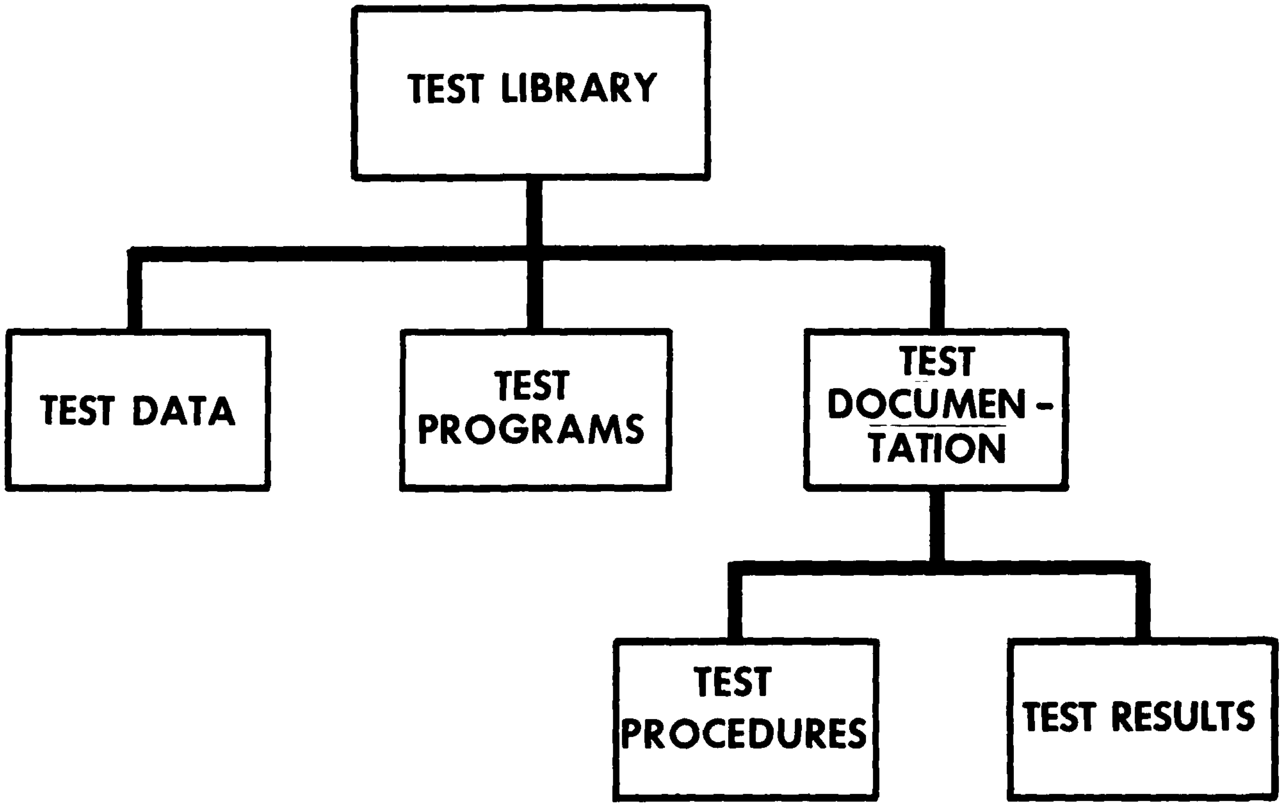
CENTACS

PROGRAM LIBRARY



CENTACS

TEST LIBRARY



CENTACS

5. Relation Between Life Cycle Milestones, Specifications, and Documentation:

5.1 General: The software and product model, defined in Section 3, provides the structure necessary to develop a maintainable system that meets its operational requirements. However, successful development of any software system depends on good management and clear definition of milestones related to the DOD Life Cycle. Two major functions must be performed to insure successful systems development. First, the combat and the material developer must jointly identify an operational or tactical need. Second, the material developer must identify progress toward the problem's solution and the method of measuring that progress. Successful performance of these functions requires exposure of software related elements as integral to the system throughout the life cycle and not as separate issues.

5.2 Specifications: Three specifications are necessary for successful software system development under the DOD Life Cycle. These specifications are functionally defined in the following sections and consist of:

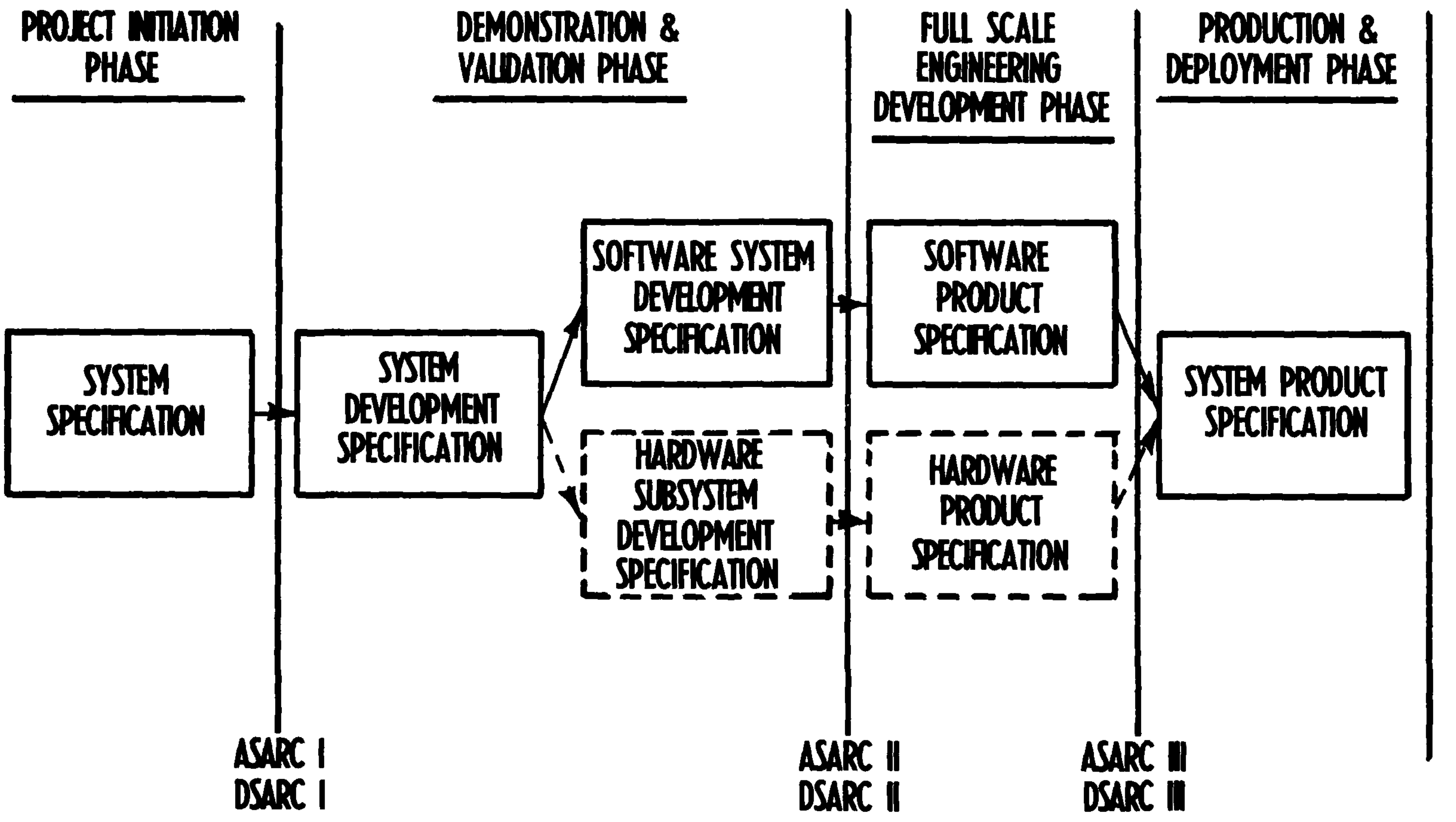
- The System Specification (Section 6)
- The Software Development Specification (Section 7)
- The Software Product Specification (Section 8)

Figure 5-1 identifies the sequence in which these documents must be developed and the time they appear in the DOD development life cycle.

The System Specification should be prepared by the material developer in cooperation with the combat developer. This specification establishes the baseline functional requirements necessary to design, test and deploy the system. Development and acceptance of this specification must be accomplished during the Program Initiation phase of the life cycle, prior to entry into the Demonstration and Validation Phase, and placed under configuration management controls.

The System development Specification is developed during the Demonstration and Validation Phase of the Life Cycle. This specification provides the allocation of functions between computer and non-computer resources within the system. It also establishes the design necessary to implement, test and maintain the software system. The System Development Specification must be developed, accepted and placed under configuration management controls prior to beginning the Full Scale Engineering Development Phase of the Life Cycle.

MAJOR SYSTEM MILESTONES



PROJECT INITIATION PHASE

DEMONSTRATION & VALIDATION PHASE

FULL SCALE ENGINEERING DEVELOPMENT PHASE

PRODUCTION & DEPLOYMENT PHASE

SYSTEM SPECIFICATION

SYSTEM DEVELOPMENT SPECIFICATION

SOFTWARE SYSTEM DEVELOPMENT SPECIFICATION

HARDWARE SUBSYSTEM DEVELOPMENT SPECIFICATION

SOFTWARE PRODUCT SPECIFICATION

HARDWARE PRODUCT SPECIFICATION

SYSTEM PRODUCT SPECIFICATION

ASARC I
DSARC I

ASARC II
DSARC II

ASARC III
DSARC III

CENTACS

The Software Product Specification is prepared during the Full Scale Engineering development Phase prior to starting low rate initial production. This specification documents the software system implementation for production and maintenance. The software product specification is accepted and placed under configuration management controls prior to production of any kind.

5.3 Specification Flow. These specifications flow one to the other. Parts of each specification will become components of the succeeding specification. The software product specification is the result of the preceeding specifications and represents the system's software product and should represent a maintainable software system that meets its operational requirements and is verifiable.

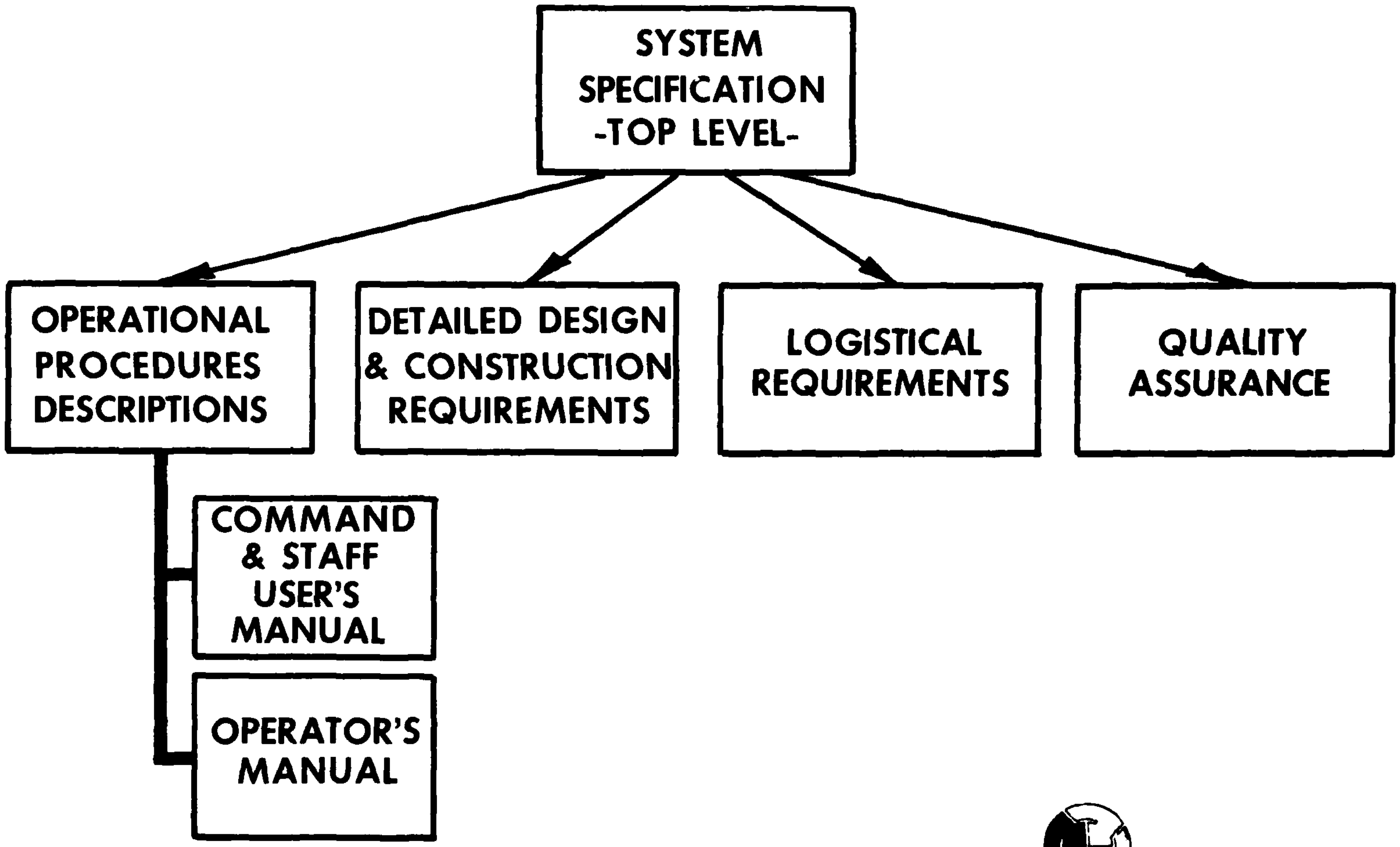
5.4 Relation Between Software Libraries and Specifications. The software specifications defined here are structured upon the elements which comprise the software end product. This structure is modular in the sense that elements of the External Documentation, Program and Test Libraries are integral parts of the specifications. As a result, the structure of documentation and specifications become congruent. Also, as prescribed in MIL-STD 490, the Development Specification (B-Level) can be discarded once the Product Specification has been completed.

6. System Specification (Type-A)

6.1 General. Upon entering the Program Initiation Phase, one of the first steps of system development is the preparation of a SYSTEM FUNCTIONAL AND PERFORMANCE REQUIREMENTS document (see Figure 6-1). This document should contain a description of the existing and proposed system as well as a summary of improvements related to the proposed system. A list of the new system impacts should be included. Based on this document, the SYSTEM SPECIFICATION, which consist of five elements (see Figure 6-2), will be developed. These elements are:

- SYSTEM SPECIFICATION - TOP LEVEL
- OPERATIONAL PROCEDURES
- DETAILED DESIGN AND CONSTRUCTION REQUIREMENTS
- LOGISTICAL REQUIREMENTS
- QUALITY ASSURANCE

SYSTEM - SPECIFICATION
(A-LEVEL)



SYSTEM SPECIFICATION -TOP LEVEL-

OPERATIONAL PROCEDURES DESCRIPTIONS

DETAILED DESIGN & CONSTRUCTION REQUIREMENTS

LOGISTICAL REQUIREMENTS

QUALITY ASSURANCE

COMMAND & STAFF USER'S MANUAL

OPERATOR'S MANUAL

CENTACS

The SYSTEM SPECIFICATION - TOP LEVEL - is a general skeleton of the overall specification of the system (in accordance with MIL-STD-490, Appendix I) containing only the top level requirements. The detailed requirements are contained in the last four documents listed above, and must be properly referenced in the SYSTEM SPECIFICATION - TOP LEVEL. The OPERATIONAL PROCEDURES document contains two parts as preliminary versions of the

- COMMAND AND STAFF USER'S MANUAL
- OPERATOR'S MANUAL

The DETAILED DESIGN AND CONSTRUCTION REQUIREMENTS include detailed requirements of system performance, design and construction. The LOGISTICAL REQUIREMENTS contain initial considerations of logistic problems. The QUALITY ASSURANCE document contains the anticipated quality control requirements, including testing. The following paragraphs present general guidance for the preparation of the SYSTEM SPECIFICATION.

6.2 System Specification - Top Level. The outline of this document is identical to the outline given in Appendix I of MIL-STD-490, but the detailed contents will be contained in reference documents described below. Refer to Figure 6-3.

6.3 Operational Procedures. Description of these documents are given in Section 9.

6.4 Detailed Design and Construction Requirements. This document, as part of the Type-A specification, shall contain all requirements which will influence the design and construction of the system to be developed. Thus it shall provide a detailed description of these requirements as called for in Section 5 of the SYSTM SPECIFICATION - TOP LEVEL. It shall contain the following:

a. Performance, design and construction requirements (subdivided into HW and SW if applicable).

b. Performance requirements related to manning and operating the system, to the extent these requirements define or constrain the design of the system.

c. Design constraints and standards necessary to assure compatability of system-HW.

d. Definition of the technical as well as the man-machine interfaces within the system and to other systems.

e. Identification and use of Government furnished property (HW and SW) to be designed into and delivered with the system, or to be used with other system equipment as an entity and an integral part of the system.

The outline of this document is shown in Figure 6-4.

6.5 Logistical Requirements. This document, as part of the Type-A specification, shall contain all requirements related to maintaining and logistically supporting the system. It will be the basis for all logistical tasks during the life cycle of the system.

The content shall be in accordance to paragraph 3.5 of MIL-STD-490, Appendix I. Additionally it shall specify the system documentation (HW and SW) (reference paragraph 3.4 of MIL-STD-490, Appendix I). The outline should be as shown in Figure 6-5.

6.6 Quality Assurance. This document, as part of the Type-A specification, shall contain all specified requirements for formal tests/verifications of system functional and performance characteristics, and operability.

The content of this specification shall be in accordance with Section 4 of MIL-STD-490, Appendix I (subdivided in HW and SW if applicable). The outline should be as shown in Figure 6-6.

7. Developing the Software Development Specification:

7.1 General. From the System Specification, Section 6, a Software Development Specification must be produced. Figure 7-1 shows the document structure for this specification. Three levels of documents exist within this structure. These are the SYSTEM DEVELOPMENT SPECIFICATION, the SOFTWARE SYSTEM DEVELOPMENT SPECIFICATION, and the SOFTWARE SUBSYSTEM DEVELOPMENT SPECIFICATION. These levels are described below.

7.2 System Development Specification. The SYSTEM DEVELOPMENT SPECIFICATION allocates functional and performance requirements to hardware and software. A brief description of both hardware and software systems shall be included in terms of block diagrams with short explaining narratives. This document should consist of only a few pages. An outline of the SYSTEM DEVELOPMENT SPECIFICATION is shown in Figure 7-2.

1. SCOPE

This specification gives a general description of the system to be developed and a sufficient overview of the other necessary documents to be used as a basis for the Demonstration and Validation Phase.

2. APPLICABLE DOCUMENTS

Only those documents referenced in Section 3, 4 and 5 shall be listed here.

3. REQUIREMENTS

Only the top level requirements should be described herein (in accordance to MIL-STD-490).

3.1 System Definition

After an overview of the functional areas and requirements of the system, the OPERATIONAL PROCEDURES and DETAILED DESIGN AND CONSTRUCTION REQUIREMENTS documents should be referenced for details.

3.2 Characteristics

Besides a summary, DETAILED DESIGN AND CONTRUCTION REQUIREMENTS document should be referenced.

3.3 Design and Construction

Besides a summary, DETAILED DESIGN AND CONSTRUCTION REQUIREMENTS document should be referenced.

3.4 Documentation

Should reference LOGISTICAL REQUIREMENTS document.

3.5 Logistics

Should reference LOGISTICAL REQUIREMENTS document.

3.6 Personnel and Training

Besides a summary, the OPERATIONAL PROCEDURES document should be referenced.

3.7 Functional Area Characteristics

General description Details in the OPERATIONAL PROCEDURES documents

3.8 Precedence

4. QUALITY ASSURANCE PROVISIONS

Besides a summary, QUALITY ASSURANCE document should be referenced.

5. PREPARATION FOR DELIVERY

FIGURE 6-3

SYSTEM SPECIFICATION - TOP LEVEL

1. GENERAL STATEMENTS

2. SYSTEM PERFORMANCE REQUIREMENTS

   2.1 Performance Characteristics (3.2.1)*

   2.2 Physical Characteristics (3.2.2)

   2.3 RAM (i.e., reliability, availability, maintainability) Requirements (2.3.3/4/5)

   2.4 Interfaces (technical) (3.1.5)

DESIGN AND CONSTRUCTION REQUIREMENTS

   3.1 Government Furnished Property (3.1.6)

   3.2 Environmental Conditions (3.2.7)

   3.3 Standard Requirements (3.3.1-6, 3.2.8, 3.2.9)

   3.4 Personnel and Training (3.6)

   3.5 Human Engineering/Man-machine Interface (3.3.7)

*The referenced paragraphs are out of MIL-STD-490, Appendix I.

FIGURE 6-4

DETAILED DESIGN AND CONSTRUCTION REQUIREMENTS

1. GENERAL STATEMENT

2. LOGISTICS

    2.1 Maintenance (3.5.1)*

    2.2 Supply (3.5.2)

    2.3 Facilities and Facility Equipment (3.5.3)

3. DOCUMENTATION (3.4)

*The referenced paragraphs are from MIL-STD-490, Appendix I.

FIGURE 6-5

LOGISTICAL REQUIREMENTS DOCUMENT

1.. GENERAL STATEMENTS (4.1)*

2. RESPONSIBILITY FOR TESTS (4.1.1)

3. SPECIAL TEST AND EXAMINATIONS (4.1.2)

4. QUALITY CONFORMANCE INSPECTIONS (4.2)

*The referenced paragraphs are from MIL-STD-490, Appendix I.

FIGURE 6-6

QUALITY ASSURANCE DOCUMENT

7.3 Software System Development Specification. The SOFTWARE SYSTEM DEVELOPMENT SPECIFICATION allocates the functional and performance requirements to software subsystems. Each software subsystem must be a separate Computer Software Configuration Item (CSCI), ref. Figure 3-2. This specification shall refer to a SOFTWARE SYSTEM OUTLINE as the overview document for the entire software system. Further, it shall refer to the detailed specification for each subsystem. This document should be brief, since it refers to subsequent documents in the structure for details. The outline for this specification is shown in Figure 7-3. An outline for the SOFTWARE SYSTEM OUTLINE is provided in Figure 7-4.

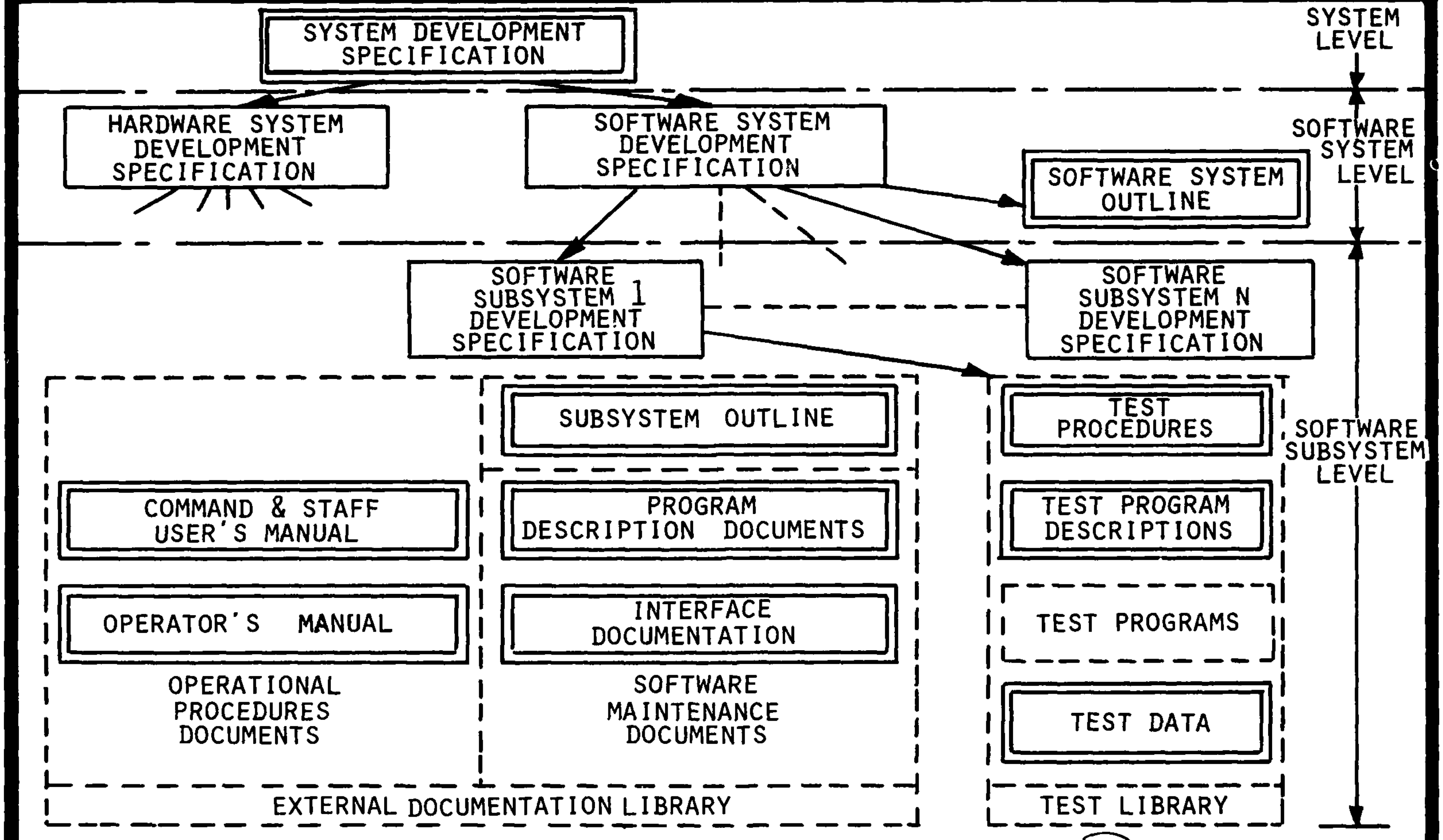
7.4 Software Subsystem Development Specification. For each defined subsystem (CSCI) there should be a SOFTWARE SUBSYSTEM DFVELOPMENT SPECIFICATION containing all requirements for that subsystem. It shall refer to the software end product documents described in Section 6, except the Program Library and the Test Programs and Test Results. The outline for the SOFTWARE SUBSYSTEM DEVELOPMENT SPECIFICATION should be in accordance with MIL-STD 490, Appendix VI referencing the corresponding software end product documentation for detailed descriptions. The documents in Figure 7-1 marked by double block are those which will be maintained throughout the whole life cycle. Refer to Figure 7-5 for the outline of this specification.

8. Developing The Software Product Specification:

Proceeding from the approved Software Subsystem Development Specification, the Software Product Specification will consist of the documents developed in Section 7. However, it will be completed by adding the Program Library (see Section 4, Figure 2-4) and the Test Programs and Test Results as part of the Test Library (see Section 4, Figure 4-5). Thus, at the end of the Full Scale Engineering Development Phase, the Software Product Specification consists of the following (see Figure 8-1):

- System Product Specification
- Software System Outline
- For each software subsystem
    * External Documentation Library
    * Program Library
    * Test Library

DEVELOPMENT SPECIFICATIONS



SYSTEM DEVELOPMENT SPECIFICATION

HARDWARE SYSTEM DEVELOPMENT SPECIFICATION

SOFTWARE SYSTEM DEVELOPMENT SPECIFICATION

SOFTWARE SYSTEM OUTLINE

SOFTWARE SUBSYSTEM 1 DEVELOPMENT SPECIFICATION

SOFTWARE SUBSYSTEM N DEVELOPMENT SPECIFICATION

COMMAND & STAFF USER'S MANUAL

OPERATOR'S MANUAL

OPERATIONAL PROCEDURES DOCUMENTS

SUBSYSTEM OUTLINE

PROGRAM DESCRIPTION DOCUMENTS

INTERFACE DOCUMENTATION

SOFTWARE MAINTENANCE DOCUMENTS

EXTERNAL DOCUMENTATION LIBRARY

TEST PROCEDURES

TEST PROGRAM DESCRIPTIONS

TEST PROGRAMS

TEST DATA

TEST LIBRARY

SYSTEM LEVEL

SOFTWARE SYSTEM LEVEL

SOFTWARE SUBSYSTEM LEVEL

CENTACS

1. SCOPE/PURPOSE

2. ALLOCATION OF REQUIREMENTS

3. DESCRIPTION OF THE HARDWARE SYSTEM

    3.1 <u>Block Diagrams</u>

    3.2 <u>Narratives</u>

4. DESCRIPTION OF THE SOFTWARE SYSTEM

    4.1 <u>Block Diagram</u>

    4.2 <u>Narrative</u>

FIGURE 7-2

OUTLINE OF SYSTEM DEVELOPMENT SPECIFICATION

1. SCOPE/PURPOSE

2. ALLOCATION OF REQUIREMENTS

3. SOFTWARE SYSTEM STRUCTURE
   (shall refer to SOFTWARE SYSTEM OUTLINE)

FIGURE 7-3

OUTLINE OF SOFTWARE SYSTEM DEVELOPMENT SPECIFICATION

1. DESCRIPTION OF OVERALL SOFTWARE SYSTEM

    1.1 Block Diagram

    1.2 Narrative

2. DESCRIPTION OF SOFTWARE SUBSYSTEMS

⋮

    2.N Subsystem N

        2.N.1 Block Diagram

        2.N.2 Narrative

⋮

FIGURE 7-4

SOFTWARE SYSTEM OUTLINE

1. *SCOPE*

2. *APPLICABLE DOCUMENTS*

Only those documents referenced within the specification should be listed.

3. *REQUIREMENTS*

3.1 Subsystem Definition

This paragraph shall provide a short description of the major functions of the subsystem. For detailed description it should refer to the Subsystem Outline and to the Operational Procedures Manuals.

3.2 Detailed Functional Requirements

This paragraph shall contain a list of all detailed requirements for each subsystem function in terms of

- *input*
- *processing*
- *output*

The detailed design description is subject of the Software Maintenance Documents.

3.3 Adaptation

Besides a short`summary the corresponding end product documents should be referenced.

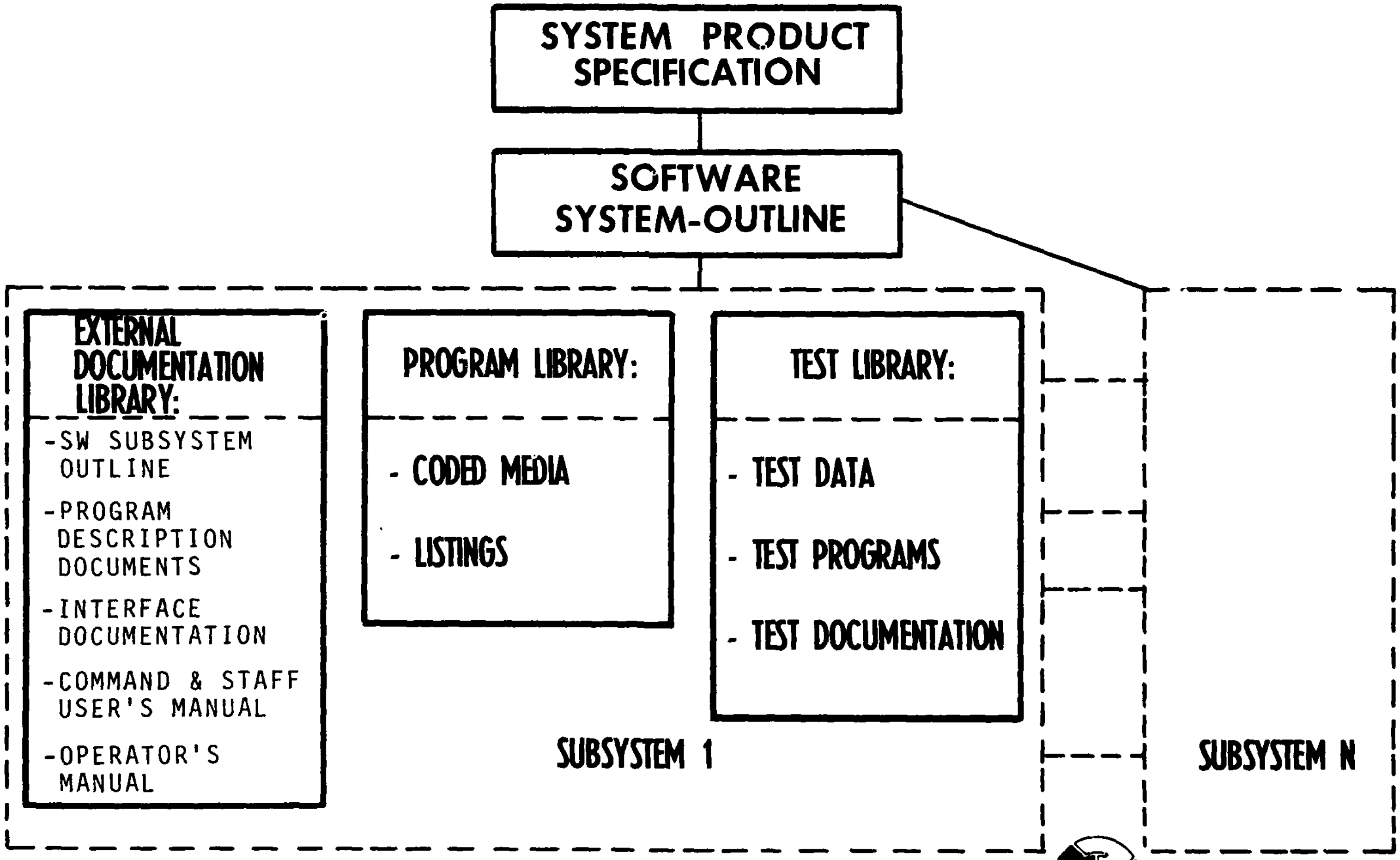
4. *QUALITY ASSURANCE*

Besides a short summary the Test Library should be referenced.

FIGURE 7-5

SOFTWARE SUBSYSTEM DEVELOPMENT SPECIFICATION

SOFTWARE PRODUCT SPECIFICATION



SYSTEM PRODUCT SPECIFICATION

SOFTWARE SYSTEM-OUTLINE

EXTERNAL DOCUMENTATION LIBRARY:

- -SW SUBSYSTEM OUTLINE
- -PROGRAM DESCRIPTION DOCUMENTS
- -INTERFACE DOCUMENTATION
- -COMMAND & STAFF USER'S MANUAL
- -OPERATOR'S MANUAL

PROGRAM LIBRARY:

- - CODED MEDIA
- - LISTINGS

TEST LIBRARY:

- - TEST DATA
- - TEST PROGRAMS
- - TEST DOCUMENTATION

SUBSYSTEM 1

SUBSYSTEM N

CENTACS

The approach taken above eliminates the Software Development Specification upon approval of the Product Specification. This is because it is totally contained within the Software Product Specification. This approach, which is consistant with MIL-STD-490, provides a well structured system of documentation and tests for quality control of the maintenance process within the product specification.

9. Supporting Standards:

9.1 General. Good management control is best maintained through the use of a set of standards which support the entire life cycle. To accomplish this, a set of integrated standards have been outlined. These standards are "integrated" in the sense that they all relate directly to the life cycle milestones, tasks and activities set forth in existing policies and procudures and provide a verifiable thread for system development. The following paragraphs contain brief descriptions of possible standards for use. Individual standards must be developed by the Army and used as needed on a project by project basis.

9.2 External Documentation Standards. Documentation is the key element for defining user oriented software systems and providing independence from the original authors. The purpose of these standards are to guide software developers in providing clear and concise descriptions of the system and individual programs as they relate to the end user, operating personnel and people who must maintain or enhance the sytem. Tn particular, these documentation standards are aimed at producting the following attributes in a Documentation Library.

* Provide a clear understanding between user and developer as to what the sytem is going to do prior to coding.

* Allow the end user to initiate requests and interpret his outputs without the aid of a computer technician.

* Allow operators to execute instructions from the end user without the aid of an analyst or programmer.

* Allow the system to be maintained or enhanced independent of any original authors.

* Allow other programs or systems to be integrated without assistance from the original authors.

These standards specify the structure and content of the External Documentation Library. They provide standards and guidance for developing and maintaining this library and define in detail the specific content of each document shown in Figure 4-2.

9.3 Program Standards. The purpose of these standars is to illustrate, for each programming language, the requirements and techniques for writing structured software code. The importance of developing such standards is clear. There is a great value in adherance to uniform rules so that programmers may more easily read and understand programs written by others. There is a need to develop coding techniques that reduce the complexity of programs, insure program portability from one machine to another, and to control the entire programming effort. These standards cover development and maintenance of the Program Library in Figure 4-4.

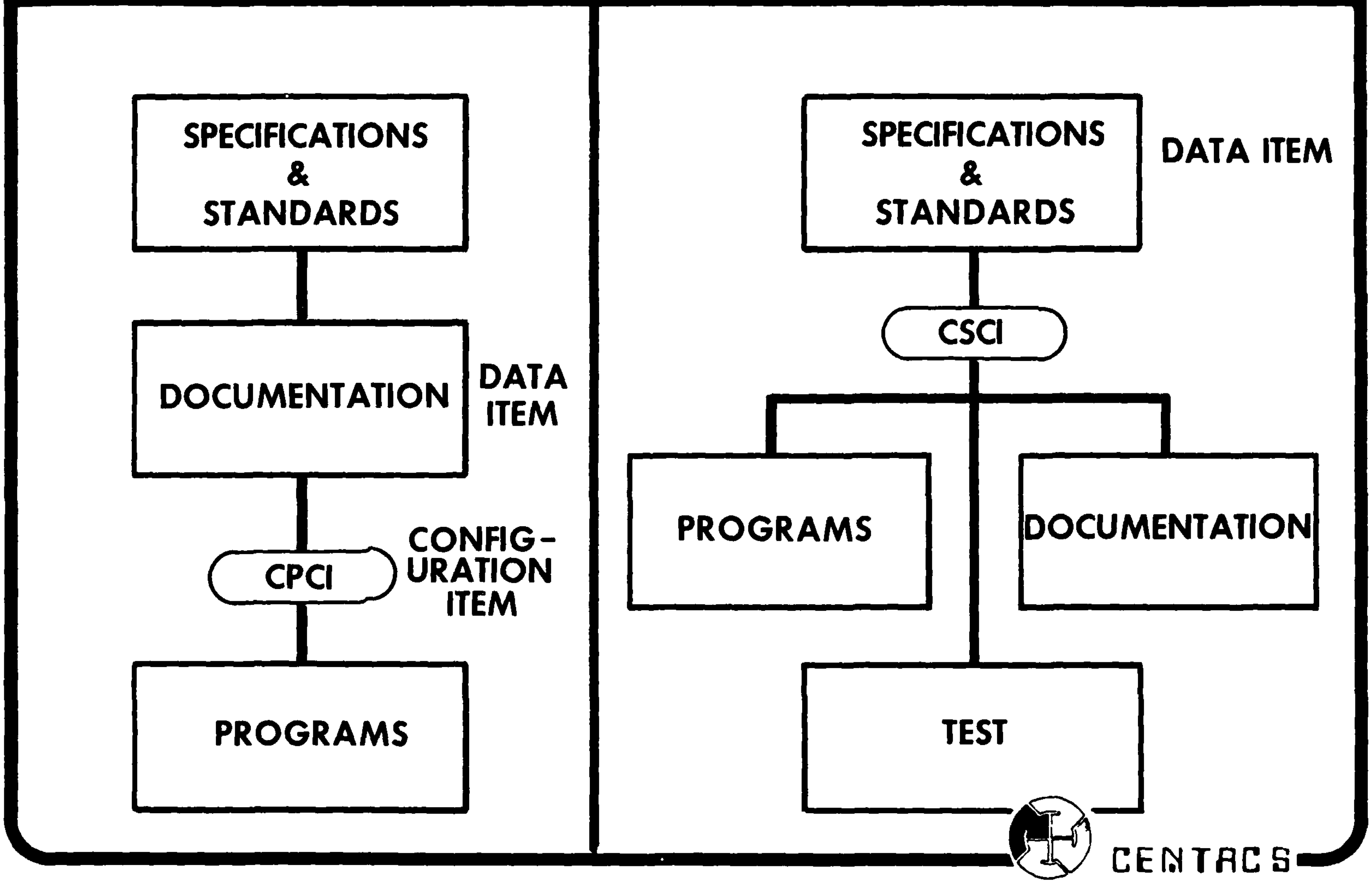
9.4 Testing Standards. The principal theme of this standard is that effective computer software testing procedures require the same level of management control as expended during preparation of the software. Software requires testing to identify faults and confirm the correctness of its performance before the system is made operational.

This standard also advances a secondary theme: test requirements can be identified as natural extensions of the system's design requirements. Also, a system's Test Library is one of the principal elements of the quality control procedural package. This standard addresses the subjects of software testing and quality control within the phase framework of the life cycle defined in Section 2. Therefore, considerable detail must be applied to management control over incremental development and use of software verification, validatability and testing to establish and maintain quality control. These standards provide guidance for the development and maintenance of the software Test Library, as shown in Figure 4-5, to be used during development and to control changes throughout the system life cycle.

10. Contractual Considerations:

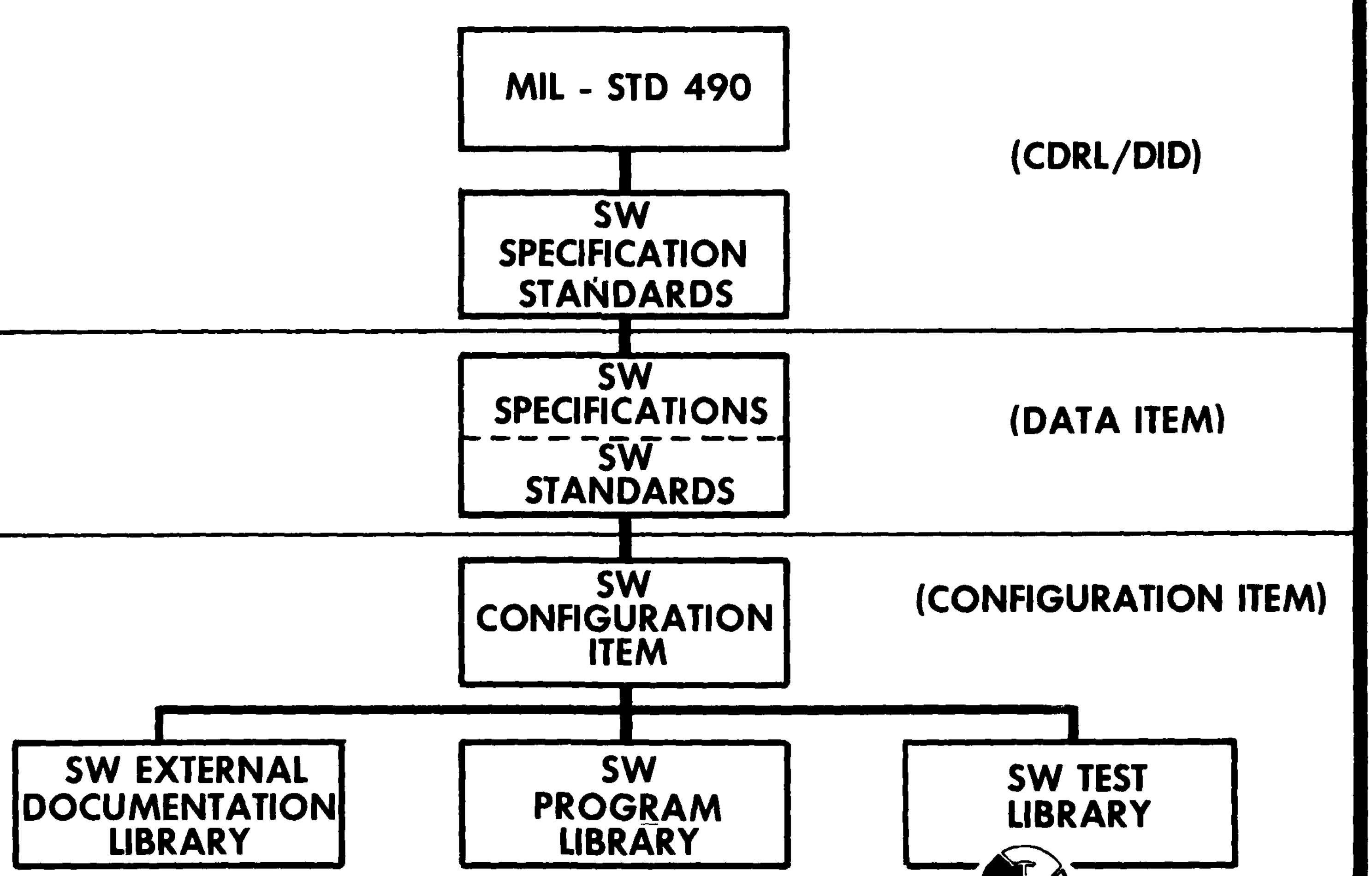
Presently, most systems under contract call for only the program library as a configuration item. As described herein, the program library is only a part of the software end product. In fact, depending on the life cycle phase, the programs themselves can be relatively unimportant compared to the documentation library. Figure 10-1 shows the present structure of contract line items, wherein documentation is a data item and only the computer programs are configuration items(CPCI's). Figure 10-2 shows the new structure of contract line items wherein the comptuer software configuration items (CSCI's) consist of all three libraries which comprise the software end product. In this case, contractor developed specifications and adopted standards are data items if necessary. Figure 10-3 shows the hierarchical structure starting with MIL-STD-490.

CONTRACT LINE ITEMS

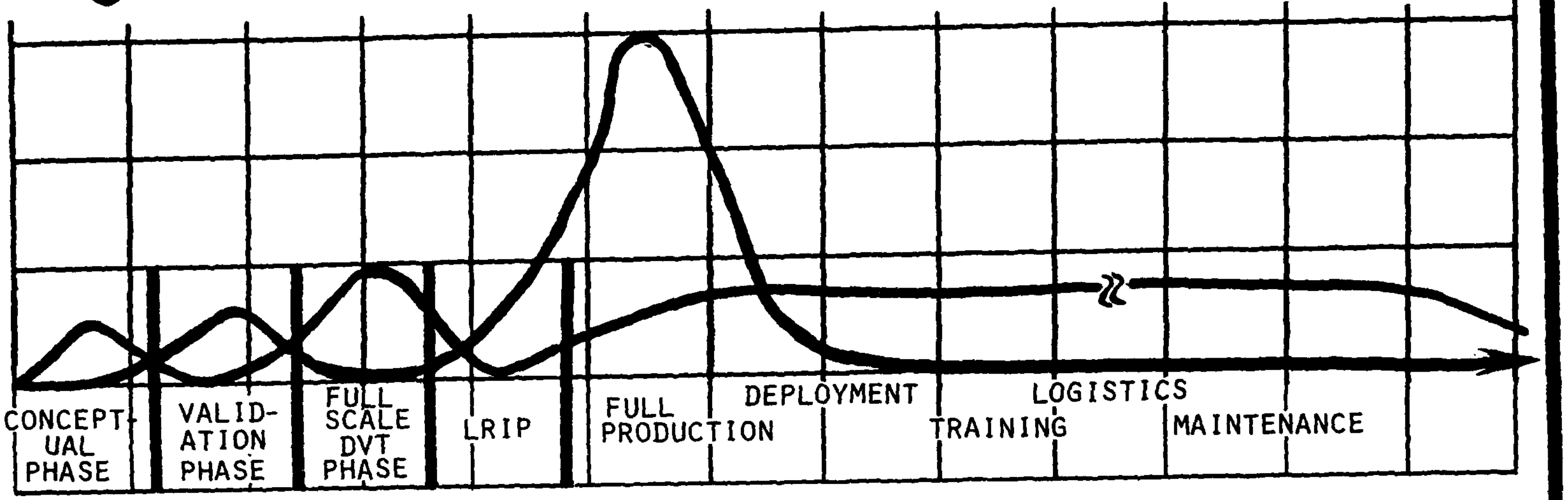


CENTRACS

CONTRACT LINE ITEMS



CENTACS



- ● PLANS
- ● SPECIFICATIONS
- ● CONTRACTS
- ● TESTS
- ● REVIEWS/AUDITS
- ● TRAINING
- ● LOGISTICS
- ● MAINTENANCE

CENTACS

Verification & Validation of Tactical Systems

LTC Charles R. Lindsey
Joseph W. D'Oria

Product Assurance & Test Directorate CORADCOM

The purpose of this paper was to examine the roles and objectives of Verification and Validation (V&V) contractors as they are applied to tactical, computer-based systems for the military.

Subsequent to a general definition of V&V, the basic tasks of a Verification/Validation contractor were discussed from a government perspective. The key elements of verification are shown to correspond to major milestones of the software development -- requirements, design, and software code verification. The independence of the V&V effort is then analyzed through a discussion of several validation activities.

Finally, special tasks and timing considerations were examined to show the extent of the V&V effort over the total software development.

VERIFICATION AND VALIDATION OF TACTICAL SYSTEM

LTC Charles R. Lindsey

INTRODUCTION

The evolution of large scale, complex, digital computer systems has recently undergone a significant period of growth. With this growth has come numerous applications of these systems to real time, command and control and tactical data systems found within the military. Unfortunately, the Government has too often experienced difficulties in developing and using such systems due to the poor quality of the associated software, the heart of any computer-based system. Numerous schemes have been proposed to prevent the generation of noncompliant, poor quality software. One of the most effective of these is the application of verification and validation (V&V) practices to the software development.

Verification and validation (V&V) is a continuing evaluation process that assures an orderly development activity throughout the software development cycle. Briefly, verification is defined as the independent evaluation process designed to ensure the consistency and completeness of the software product at any given phase within the development cycle. The consistency aspect is concerned with measuring the degree to which a given phase (e.g., design) is in agreement with the previous phase (e.g., requirements) in the development cycle. Completeness is a measure of the readiness to initiate the next phase in the development cycle. Validation, on the other hand, is oriented toward the final software product; it is directed at software test and evaluation to measure how well the software executes according to established requirements.

Significant cost savings and increased confidence in a software product can be realized through the support of a well-qualified V&V contractor during all phases of development for tactical systems.

Such support is designed to provide in-depth technical visibility that is essential for the Government to maintain control of the overall software development and to contribute significantly to the confidence in the program's success. V&V should emphasize the significances of quality specifications, sound standards and procedures, effective configuration management, comprehensive design reviews, quality test planning and documentation, independent validation, and a professional diplomatic technical interface with all parties involved.

To be truly effective, a V&V contractor must have the skill and diplomacy to communicate information concerning problems and potential solutions in such a way that the progress of the system development is enhanced. By helping to solve problems, a V&V contractor becomes a viable asset to ensuring a quality software product and not another critic (in a negative sense) that only points out how bad things are.

With this introduction to software V&V, we will examine the role of the V&V contractor in terms of general activities shown in Figure 1.

VERIFICATION

The following verification activities are considered the most critical in assuring an orderly software development.

Requirements Verification

One of the keys of a successful V&V contractor is the ability to focus on inadequacies and problems in software requirements. This ultimately will lead to the identification of potential problem areas early in the development cycle. Complete and unambiguous software requirements are a necessary foundation for software development of large complex tactical systems and one standard for measuring software quality. A V&V contractor should be able to perform an exhaustive evaluation of the requirements to measure qualities such as testability, traceability to higher specifications, completeness, adequacy, and degree of ambiguity that may exist in the system/ software specifications.

Clearly, the specifications are crucial in the development of the software because they define required data processing to be accomplished in the overall tactical application. Therefore, the V&V contractor should analyze and evaluate the system specification and the software requirements to verify the testability and implementation attributes required by the software development process. An important side benefit of this analysis is the support of independent test activities by providing the relevant objectives for candidate independent tests.

Figure 2 shows a representative list of criteria that should be used in the evaluation of these documents. Two of the evaluation criteria stand out as being of major importance to software V&V: uncover implicit requirements and determine whether requirements are testable. Based on past experience, these two criteria are the most important to understanding both the software and the test data that results.

This evaluation also represents the first phase of an incremental technique for verifying that the code correctly implements the requirements as stated in the system/software specifications. A V&V contractor should trace each requirement to successively lower documentation levels until the lowest level software design documents are traced into the actual code. The approach is based upon the relationship between documents of the hierarchy in which each requirement at one level must be logically traceable to requirements in the next higher level and vice versa. A V&V contractor must recognize the evolutionary character of most tactical programs and stand ready to evaluate major revisions and/or updates to the system/software specifications as they occur.

Requirements analysis is directed at resolving requirements issues/problems to ensure that the following characteristics are realized.

Realistic - Requirements must be achievable within the capabilities of the data processing hardware.

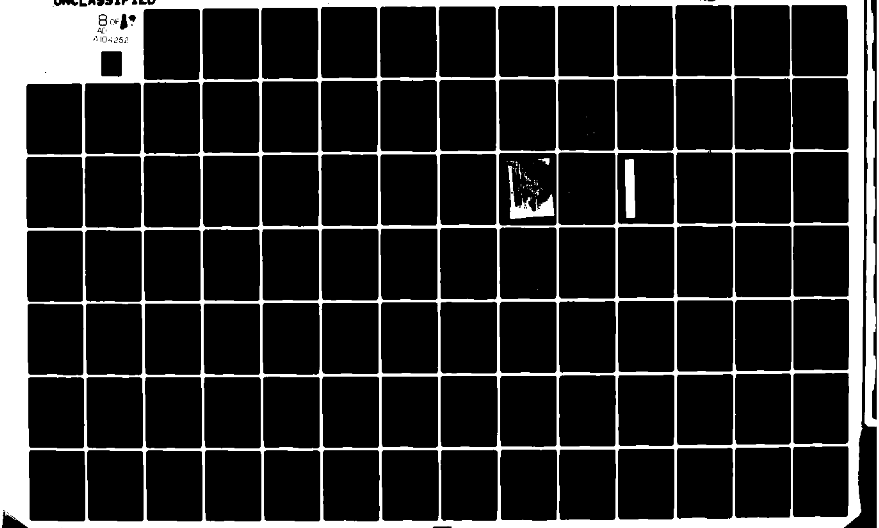
Unambiguous - Requirements must be stated such that they are definitive and not open to subjective interpretation.

Consistent - Requirements must be consistent with one another, with interfacing subsystems and with those at the next higher and lower levels.

AD-A104 252 INTERNATIONAL BUSINESS SERVICES INC WASHINGTON DC F/G 9/2
US ARMY SOFTWARE SYMPOSIUM (2ND) HELD AT WILLIAMSBURG, VIRGINIA--ETC(U)
1978 S M TAYLOR DAAK70-78-D-0030

UNCLASSIFIED NL

8 OF 9
AD
A104252



Necessary - Requirements must be necessary. Unnecessary or overly restrictive requirements will increase the cost and complexity of the software and will also impact the design code and testing schedules.

Complete - The requirements must completely specify the software product to be provided in terms of accuracy, timing, throughput, interface control, reliability, maintainability, environmental conditions, input/output, and human factors.

Testable - The requirement must be testable.

Design Verification

Before coding begins, the V&V contractor should verify that the software design accurately reflects software functional and performance requirements. This is important to reduce the risk of entering into an expensive cycle of specification design changes which would ripple throughout the programming activities. Although some assurance of design soundness can be gained from verifying the flow charts and design descriptions by themselves, the tracing of the design to requirements provides maximum payoff. This encompasses an exhaustive examination of the documentation that includes a step-by-step comparison of the flow charts to the higher level documentation.

An approach to design verification consists of a detailed evaluation to ensure:

Adherence to flow chart standards to reduce the possibilities of misinterpretation because of variations in the use of flow chart symbols;

- That program flow charts describe the flow of data and control through all parts of the software system;
- That emphasis is placed on the hardware involved, data transformations made, and the work stations through which control passes; and
- That the program flow chart is a detailed graphic representation of the specific operations and the logical sequence required by the software requirements.

Lower level flow charts should also be evaluated to ensure that they contain detailed graphic representations of the specific operations to be performed within the individual program components and the logical sequence in which they are to be conducted. A V&V contractor should ensure that the flow charts from one level to the next are consistent and that the lowest level flow charts are adequate for initiating the next phase of coding the software. In addition, the following should also be verified:

- Timing requirements in the lower level flow charts are consistent with the allocations from the higher level;
- The internal/external parameters and file names adhere to the data base design specifications;
- Linkage arguments are consistent between calling and receiving modules; and
- Dependency relationships between modules are consistent and satisfy upper level requirements.

In addition to the specification analysis described previously, the V&V contractor should analyze the logic associated with each of the functional algorithms. This may be done by selecting a group of critical functions, coding and executing them on the V&V contractor's computer. A driver can be designed to test the coded functions, and the results can be checked against expected results derived from flow chart descriptions. Such an approach ahs been used to evaluate algorithm completeness, data base adequacy, and overall control logic within the individual module. These coded routines may also be used later in the test verification activities.

After completion of the flow chart verification, the V&V contractor should continue to review design changes throughout the remainder of the development cycle. This procedure is necessary to maintain quality documentation for later phases.

Critical problems discovered and corrected during the software design phase will eliminate significant cost schedule impacts. Experience has shown that design problems that are not uncovered until late in the development cycle are not only costly to correct, but also contribute to a lower quality in the software product. V&V contractor activities in the past have contributed significantly to the development of lower cost, high-quality software through elimination of a majority of the design problems early in the development cycle. Problems discovered during the design phase are easily corrected, whereas those discovered during the testing cycle most often require an expensive modification of design specifications and code and usually require that portions of the software tests be repeated.

Software Code Verification

Independent verification of the coded software is essential to assure consistency and completeness between the code and the design. The V&V contractor should verify the assembly and/or HOL source code to determine that:

- The code satisfies the intent of the previously verified flow charts;
- The code contains comments adequate to support software maintenance and test;
- Military coding standards and procedures are adhered to; and,
- The code is designed to efficiently utilize the hardware features of the host processor.

This step-by-step verification procedure is designed to ensure that if the code satisfies the intent of the flow charts, it will satisfactorily implement all of the requirements of the software specification. The V&V contractor should verify the code against the flow charts by checking the comments in the code to ensure that they agree with the commentary in the flow charts. In addition, the V&V contractor should check all interfaces, ensure that all branches have been properly coded, and check the coding of all mathematical transformations.

VALIDATION

So far the discussion has addressed verification efforts only. Some of these efforts overlap validation activities. The distinction between the overlapping V&V tasks should become obvious as validation is discussed.

Validation comprises those evaluation, integration, and test activities carried out at the software system level to ensure that the final developed software satisfies the requirements of the software system specification.

The validation activity parallels testing and integration phases of the software development. It can begin when sufficient amounts of code are present to perform validation of functional and performance requirements.

Validation activities can be segmented into three tasks, each of which addresses a separate phase of the validation effort. The first task involves the monitoring and evaluation of the prime contractor's test activities. This will allow the Government to be continuously aware of the validation progress and status. The next task is the independent execution by the V&V contractor of selected prime-contractor-designed validation tests in order to audit the repeatability of validation results. The last task is V&V-contractor-executed validation tests to test cases created exclusively by the V&V contractor's own analysis to extend the preexisting validation results beyond that conducted by the prime contractor.

Validation of Contractor Assurance Activities

The V&V contractor should monitor the prime contractor's software assurance activities to enhance the Government's visibility into adequacy and status of these activities. This task should be designed to evaluate the prime contractor's software testing to ascertain the degree to which the testing demonstrates the software's compliance with Part I Specifications.

As a result, the Government will benefit from an up-to-date, independent interpretation of the ongoing testing efforts plus an objective appraisal of the thoroughness and rigor with which the software is validated.

The V&V contractor should have access to the prime contractor's validation documentation, including test plans, test procedures, and test execution reports. Using these inputs, the V&V contractor evaluates the testing to ensure that:

- All validation tests are traceable in their intent to the Part I Specifications,
- All areas of the software are addressed by validation testing;
- Unnecessary redundancy between validation tests is eliminated;
- Adequate test data is recorded to evaluate the results of the test; and
- Critical software components are appropriately stressed during validation; and
- Test data is correctly interpreted and applied to derive final test results.

In addition to evaluating the validation documentation, the V&V contractor may be tasked to monitor integration/validation demonstrations and formal/informal reviews. In addition, he may be expected to critique current test plans, procedures, and reports. The current test schedules should also be monitored providing advanced warning of any potential slippages.

These efforts will allow an objective, informed evaluation of the prime contractor's validation program and provide independent review of the testing status.

Selective Testing

The independent audit of selected prime contractor tests by the V&V contractor can help to assure the integrity of software compliance with regard to Requirements Specifications. The direct involvement of the V&V contractor with the execution of contractor-developed software will demonstrate the learnability and transferability of the software product. This hands-on experience will give the V&V contractor an opportunity to independently evaluate the man/machine user interface designs. It also offers the V&V contractor an opportunity to understand the testing philosophy and procedure of the system; this facilitates the generation and conduction of separate validation tests by the V&V contractor at a later phase.

To accomplish this audit, the V&V contractor will require access to the baselined, configuration-managed contractor-developed software; periodic blocks of available computer time on a suitable facility; and various support software packages and prime contractor validation test documentation. By using this input in conjunction with the evaluation results of the previous activity, the V&V contractor is able to select a meaningful subset of prime contractor validation tests for analysis. This meaningful subset should be comprised of those prime contractor tests which which are technically crucial to the functioning of the entire systems under development. Thus, tests that exercise key algorithms, mode changes, error responses, and queue/buffer/stack limits, or which stress compliance with critical timing or throughput requirements should be candidates for revalidation.

Tests selected for revalidation must be well-documented, and the inherent randomness of software response must be low so that test repeatability is feasible. The V&V contractor can then attempt to duplicate the results of each selected test by the prime contractor by first assembling the particular test inputs, including preset data such as cards or tap and any manual real-time inputs that may be specified in test procedures.

After obtaining and recording the test data results, the V&V contractor can compare his independently obtained results with those obtained by the software developer and can identify any differences that may exist. It is this phase in which the V&V contractor's experience plays a vital role in judging how potentially controversial test result mismatches should be handled. Close communication between the Government, the prime contractor, and the V&V contractor is essential to quickly resolve the differences to prevent animosities and prejudices from developing.

Independent Validation

Independent validation of the tactical software by the V&V contractor can establish additional confidence in the quality of the tactical software by extending the validation boundaries. After having performed the audit of prime contractor validation, described in the previous task, the V&V contractor should be able to rapidly and inexpensively validate the tactical software in areas of design and to limits of stress not achieved in earlier tests.

The Independent Validation task has two primary objectives, the first of which is to validate the tactical software to baselined test requirements that may be beyond the scope of prime contractor validation because of resource limitations or oversights. The second objective is to extend the results of prime contractor validation into additional scenarios and test cases. Although these tests will not necessarily extend software validation to address any additional requirements, it will demonstrate requirements compliance for a more diverse set of data conditions than those specified in the prime contractor's validation.

Experience has shown that errors discovered during the course of independent validation are found at greater rates when the independent validation places greater stress on the software than is normally applied during the original validation.

ADDITIONAL TASKS AND FACTORS

Evaluate Test Plans and Procedures

Exhaustive testing at each level of development must be enforced to assure that the operational software will satisfy the requirements and design specification. During the testing cycle of the proposed contract period, the V&V contractor should evaluate the test plans of the development contractor, recommend test tools and techniques which will contribute to increased confidence in the software, and perform independent testing on critical software items as necessary. Also, the contractor's testing cycle should be monitored to ensure that test plans have been followed and that the test results satisfy preestablished acceptance criteria and have been properly documented and interpreted. Test plan documentation should be analyzed to assure that all requirements are addressed, that all tests exercise the code adequately, and that test tools and test procedures are utilized to minimize the occurrence of potential problems.

In addition to the evaluation of test plans and procedures, the V&V contractor should independently evaluate the software developer's test reports to determine such items as percentage of code and interfaces exercised, and acceptability of test results.

Special Tasks

The V&V contractor is often requested to conduct special tasks such as performance analysis studies that provide technical responses to critical, high-risk areas which arise during tactical software development. The ability of the V&V contractor to provide special software performance-related studies offers the Government the capability to more easily adapt to problems that were unanticipated at the outset of the design effort. In the development of any large-scale software program, there will inevitably be inconsistencies, design flaws, and other shortcomings in the software which could not have been predicted at the outset of the software implementation effort.

The types of studies envisioned center around potential system-related problems. Examples of such types of studies that are often performed in a V&V effort include CPU overload considerations, port-to-port timing analyses, worst-case stressing scenario development, and unit sizing. The V&V contractor should be able to develop, modify, and apply special-purpose models and simulators to evaluate tactical system and function performance. Such models may be used to derive pertinent statistics regarding throughput of critical threads, growth capabilities (timing and storage), I/O degradation performance, bottlenecks, etc.

Timing Considerations

The V&V contractor should be "on board" at the beginning of the development cycle. If the V&V contractor is under contract as early as the development of the RFP (prior to the proposal period), significant recommendations can be made to the Government concerning Contract Deliverable Items, design reviews, configuration management, visibility to the Government during the development cycle, etc.

It is recognized however, that many programs are currently under development. Benefits can also be realized by the Government in this instance even late in the software development cycle. Activities relating to 1) evaluation of the software developer's test plans, procedures, and results data, 2) independent exercise and evaluation of the software product by the V&V contractor, and 3) preparation of an independent agency to support software maintenance activities can contribute significantly to "measuring" the quality of "as-built" software and recommending corrections for problems uncovered.

For programs that are currently within the concept and requirements definition phase, it is recommended that a full range of V&V activities be applied, from early requirements analysis through final independent system performance validation. This range includes tasks to assure that the requirements are properly allocated throughout the design and code, evaluation of design specifications, evaluation of test planning by the software developer, evaluation of simulation and system test results, and special task to address proposing solutions to key problem areas in which the Government desires an independent recommendation.

For software projects that are currently within the design phase, it is recommended that emphasis be placed on the evaluation of the design specifications to determine their consistency and completeness, and also analysis of the traceability between requirements and design specifications. Once the adequacy of the design has been assured, the code should be verified against the design documents to ensure that the design has been implemented faithfully and efficiently.

For software projects in the final stages of integration and test, it is recommended that the V&V contractor first evaluate the prime contractor's test documentation to assess the need for additional testing and to evaluate test results. Second, the V&V contractor should exercise the completed software over a wide range of input conditions to independently evaluate both functional and performance characteristics. Third, the V&V contractor should conduct evaluations of as-built documentation to determine its adequacy to support the operation and maintenance of the software to be delivered to the Government.

SUMMARY

From the above, one can appreciate that the wide range of potential activities and benefits to be realized by making effective use of a qualified V&V contractor. Such a contractor can be effective for software systems at any stage of development; however, the maximum benefit to the Government can only be gained by initiating the V&V effort as early in the development cycle as possible. The exponential costs of correcting errors committed early but only found late in the development cycle provide economic justification for early participation of the V&V contractor.

Software Testing at the System Level

J. Gary Nelson

Headquarters
U.S. Army Test & Evaluation Command

The paper provided five principles that should be observed in establishing a viable software testing program (to include system level testing). Strategies were examined of the integration and system levels. Test methodologies of vital importance to the system tester were provided. Among these were software requirements identification and tracing, testing to requirements, software/computer system simulation and instrumentation.

SOFTWARE TESTING AT THE SYSTEM LEVEL

Mr. J. Gary Nelson
US Army Test & Evaluation Command

INTRODUCTION

A French meteorological satellite sends erroneous destruct signals to 72 of 141 high-altitude weather balloons. (2)

The Ballistic Missile Early Warning System (BMEWS) mistakes the rising moon for a massive Soviet missile raid. (3)

Both problems occurred because of computer software shortcomings. Consider the Strategic Air Command's Automated Command Control System (SACCS 465L). One software error happened each day. About 95% of the SACCS 465L software delivered to SAC had to be rewritten. (1)

Why did these critical problems go so long without discovery and correction? Because the state-of-the-art in software testing lacked the sophistication to uncover them during the development test process; because proper testing that may have uncovered them early in the development cycle was either done poorly or not done at all.

The last 20 years have seen unprecedented technological advances in reduction of size and weight in computer hardware, plus improvement of the computer power per dollar. These advances have spurred the technical community into finding more and bigger jobs for the cheaper resources to do. The technology has gone from tubes/transistors through integrated circuits to medium-and large-scale integrated elements. However, two decades ago, software was in an era when highly efficient programs which used little of the scarce and expensive computer hardware resources were sought ...when the size and complexity of the programs could be handled by one good programmer ... when the design of the software was largely left up to the programmer's art ... when he tested his own product. We now see software moving toward an era when high efficiency is not the prime concern since computer hardware resources are not scarce or expensive ... when the software design is extremely complex ... when self documentation and readability of the code demands programming standards rather than individual programming style ... when the only thing that programmer testing will show is that it works exactly as he programmed it.

From an economic perspective, it is predicted that in 1985, 95% of the total system development costs will be allocated for software. (1) If one combines these estimates with the fact that 34 to 50% of software development costs are devoted to checkout and test, it becomes obvious that software testing merits considerable attention.

This article examines the complex processes of software testing emphasizing some of the more critical issues.

TESTING PRINCIPLES

Before we discuss specific testing methodologies, let us state some general philosophical principles that a developer or project manager must consider in establishing a viable and profitable test program. Some of these issues will be reexamined in more detail later in this chapter.

Integrated Test Planning

Software testing is not a one-time event. Adequate testing can only be achieved if it is performed throughout the development phase (and, to some extent, during the operational phase) of the system life cycle.

As was mentioned before, software problems still exist in often aggravating quantities through deployment and into operational utilization. Therefore, testing should be considered throughout the system's useful life. Please note that testing here connotes planned, disciplined, instrumented, repeatable exercises of the system or some developmental test bed (or the requirements and specifications themselves) with the expressed purpose of determining some attribute or performance parameter of the computer/software. Operational testing and field utilization of the system should not be included even though problems may be uncovered there. There are at least four general levels at which software testing should be considered.

The development of a computer based system must include all of the participants, especially the testers, from beginning to end. In order to provide the required lead time and prerequisite activities needed to assure smooth development and testing, a group which might be called a computer resource working group should be formed early from representatives of the user, tester(s), contractor, and project manager's office. This should be the forum whereby inputs from the various disciplines can be presented, integrated, and (if necessary) traded off to the benefit of the project as a whole.

Testing Organizations

Usually, the developing contractor's job is to deliver a complete system; so, depending on his facilities and capabilities, he could provide testing at all levels. In the late 50's the Atlas Missile Program hired an "Independent Software Tester" to provide additional unbiased software test support. This Independent Software tester concept has since been used on nearly 20 major defense and NASA systems. Now called a Validation and Verification (V&V) contractor, his job extends over the module, functional area, software system, and in some cases, target system testing.

Regardless of the names of the tester(s) the project manager must assure that the software testing is coherent and flows smoothly through the various hierarchical levels. This means that sufficient information need be transmitted to and among all testing participants throughout the entire development cycle and that each tester knows explicitly how he fits into the testing scheme.

Because this testing does exist throughout several phases of the life cycle, different organizations institute various types of testing each with differing methodologies and goals. The programmer performs debugging and checkout; quality organizations perform QA testing or V&V testing; other groups are responsible for evaluation testing. While still other groups may do acceptance testing. The point is that since various organizations or activities do perform testing, it is difficult to single out one individual or organization which has the total responsibility for all testing functions.

One wcrd on independence of the testing organizations. As previously mentioned, the early V&V testers were called "independent testers"... but, independent of whom? They worked for the project manager, not the contractor. This is not to imply that the contractor would be dishonest. It does imply that the creator of a software product is naturally biased toward the way he produced it. A programmer, testing his own product will always show that it meets the requirements (otherwise, nothing would be released). As the development reaches the system integration level, the testing should be even more independent of the developing contractor. However, tesing can never be independent of the project manager. To reemphasize, testing must be an integral part of the development process, not relegated to down-stream, poorly understood circle on a PERT chart that is destined to terminate when the budget runs out. If used wisely the payoffs of a good testing program are high.

Attitude

A proper attitude toward testing in general must be established. Software problems historically linger beyond deployment. Testing, therefore, must be directed toward the identification and erradication of software problems. This direction is inevitably at odds with the idea that a development in which few problems are evident is aesthetically better than a development in which many problems are discovered. In this type of system development the opposite is true. However, from a testers point of view, he is hated most when he does his job best. Additionally, the tester should never be used to give demonstrations and political shows, especially for computer-based systems. First, this type of system is likely to defeat the purpose of the demonstration by failing in the middle of the performance; second, the tester's time and the developer's usually scarce money can be better spent determining where, when, and how the code "breaks".

Prerequisite Functions

Another basic principle is to assure that testing-related prerequisite functions are done in a timely fashion. When not adequately provided for, testing becomes impossible to perform. The result is that testing is delayed and/or doesn't provide definitive information about encountered problems. Systems reach critical decision points with the software not sufficiently developed. System level testing too often becomes a software find-and-fix debugging exercise which, for computer-driven systems, is extremely expensive and very time-consuming. Testing run-ons and delays, coupled with mounting costs, pressure decision makers into fielding "something" and fixing it later, hopefully.

Just what are some of these test related prerequisite functions? Probably the single most important prerequisite function is the drafting and verification of system/software requirements and specifications. It is here that the engineering aspects of software are displayed. In this type of system the software must be designed before it is coded. This must be done in a top-down fashion to assure that what is coded satisfies the system's intended goals. Within the Defense community the hierarchy of requirements and specifications is spelled out in MIL-STD-490 and (for software in particular) MIL-STD-483. Each level is an embellishment of the one above it. In software, a high degree of explicitness is needed to assure success in not only testing, but in every facet of the RDT&E process. Good requirements and specifications are really the development's road map.

Another prerequisite function is the planning for and procuring of test instrumentation and test tools. Many automated tools exist on the open market today. However, molding and modifying them to fit the particular software at hand is not always an easy (or practicable) enterprise. These automated test tools include categorically:

- Static Analyzers
- Code Analyzers
- Symbolic Evaluation Systems
- Self Metric Instrumentation
- Dynamic Assertion Processors
- Test Data Generators

- Test File Generators
- Execution Verifiers
- Output Comparators
- Test Harnesses
- Software Monitors
- Hardware Monitors
- System Drivers

The testing level that uses the particular category of tools moves from raw code to system testing as per this list. In general, considerable lead time will be required to adapt or build these tools. For example, the system level driver is often as complex a device as the system being driven. It must be operational and validated by the time the target system is ready for system level testing.

Another prerequisite function that should be performed is simulation, especially computer/software system simulation. Computer/Software system simulations are discrete event simulations that represent the utilization of hardware resources in time during the operation of program/module code segments. Operating system functioning is also played. These simulations, originally designed and used to study data processing installation hardware and job stream architecture, are extremely helpful throughout the development of these computer-driven systems. Among their uses are:

- Sizing Studies
- Hardware/Software Trade Offs
- Timing Studies
- System Parameter Interrelationships
- Design Sensitivities
    - Resource Utilization
    - Time Marks and Windows
    - Computational Accuracies
- Test Scenario Design
- Failure Analysis
- Educational Tool

This type of simulation can easily be written utilizing several existing macro-language software packages available commercially. These packages allow the user to insert hardware and software architectural, control, and performance parameters plus a dynamic input stream. The simulation outputs can cover a wide range of resource utilization reports in terms of time accumulation and distribution. As the system software (and hardware) parameters become more firmly known, this "table-driven" simulation can be easily updated.

Since this simulation can be used by all participants in all stages of development (plus maintenance), it should be physically located centrally within the developing organization in order to be accessible to all participants.

These prerequisite functions will be addressed later under methodologies.

Additional Axioms

We add to the set of principles discussed above, additional testing axioms given by Meyers (8) which should be considered by management in formulating its software testing strategies. These axioms do not represent a complete list; and we offer them here without explanation.

- A good test case is a test case that has a high probability of detecting an undiscovered error, not a test case that shows that the program works correctly.

- One of the most difficult problems in testing is knowing when to stop.

- It is impossible to test your own program.

- A necessary part of every test case is a description of the expected results or output.

- Avoid nonreproducible or on-the-fly testing.

- Write test cases for invalid as well as valid input conditions.

- Thoroughly inspect the results of each test.

- As the number of detected errors in a piece of software increases, the probability of the existence of more undetected errors also increases.

- Assign your most creative programmers to testing.

- Ensure that testability is a key objective in your software design.

- The design of a system should be such that each module is integrated into the system only once.

- Never alter the program to make testing easier.

- Testing must start with objectives.

TEST STRATEGIES

As was previously mentioned, this article considers four levels of testing within the development effort -- module coding and debugging, module integration or functional area building, software system integrtion,and target system integration. Figure 1 depicts this graphically. The module coding and debugging level is done mainly by the programmer himself and is highly dependent upon static, syntaxical checks and is highly "answer" oriented (where the exact result is anticipated). The module integration level can be performed by programmer teams and/ or by test organizations both within and outside of the developing contractor's organization. Here, the testing is much like module testing plus the problem of module integration on a "local level." Some dynamic testing is usually introduced at this level.

After individual modules and program segments (functional areas) have been tested (debugged and validated), we are faced with two major categories of tests. The first of these is software system integration testing where the various modules and/or functional areas are merged together to form the entire computer program. Several possible approaches can be used to combine the modules each with various impacts upon testing. Software system integration testing is primarily used to validate proper interface (data flow and control) among the modules that constitute the computer program.

Once the software is integrated one then considers total system testing. System testing is not a comprehensive function test where each function described in the B level software specification is validated. Most sofware systems are too complex to be subject to such exhaustive testing after integration. Rather, system testing attempts to expose inconsistencies between the system and its original system level requirements, while at the same time investigating suspected software weaknesses. It also serves to validate "assumed" or simulated environments or inputs.

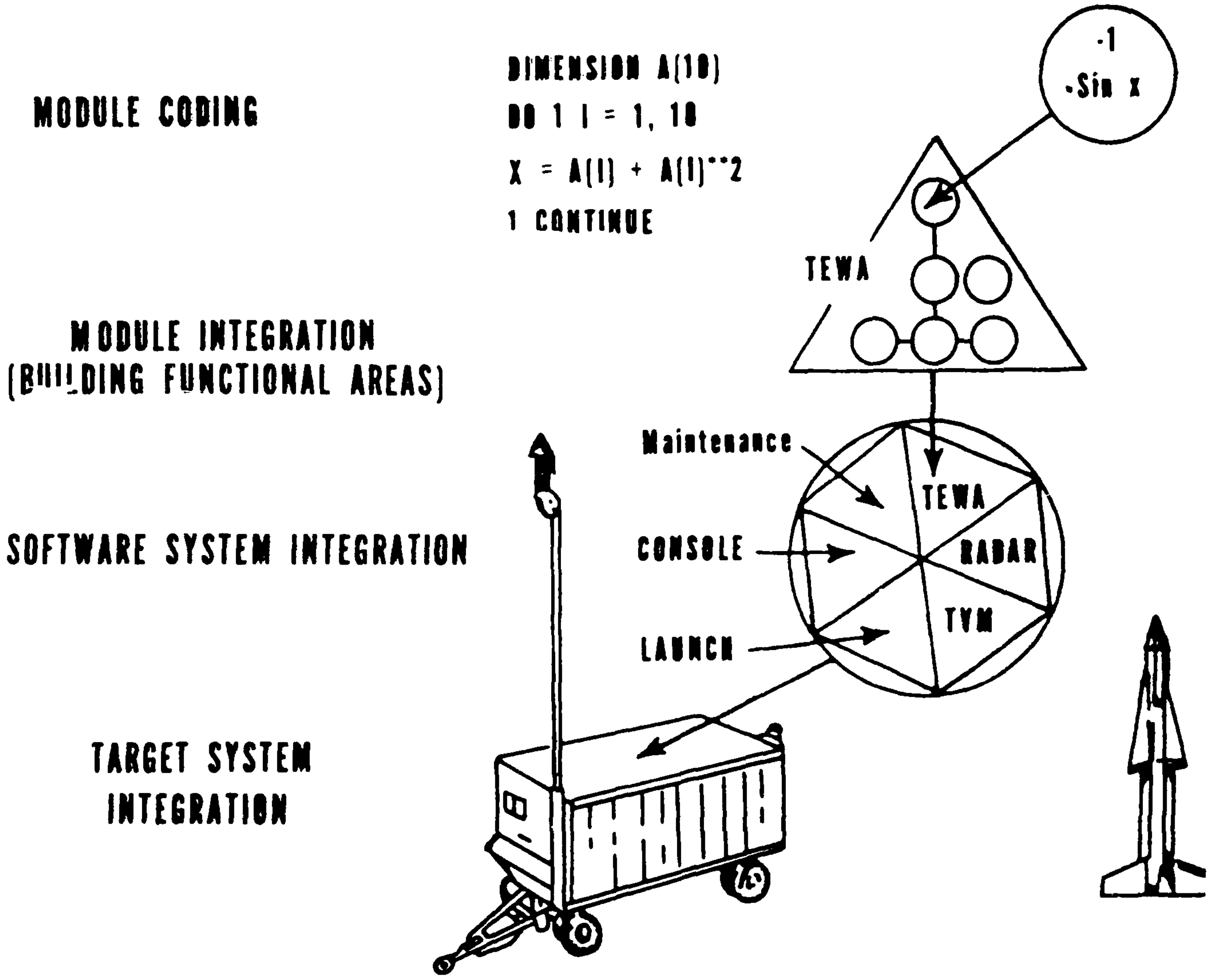
Integration Testing

The process of putting modules together to make functional areas and putting functional areas together to make software systems is known as integration. The testing here is directed at the interfaces and the abilities of mutual coordination and tolerance among the code segments.

Classically, there are two types of integration testing: bottom-up and top-down. However, more recently variations of these strategies have emerged. These variations attempt to utilize the best features of bottom-up and top-down approaches while minimizing the disadvantages.

o Bottom-up testing: As the name implies, this strategy puts unit testing ahead of everything else. Modules that call or invoke no other

PHYSICAL SOFTWARE DEVELOPMENT AND TESTING LEVELS



MODULE CODING

```
DIMENSION A(10)
DO 1 I = 1, 10
X = A(I) + A(I)**2
1 CONTINUE
```

MODULE INTEGRATION (BUILDING FUNCTIONAL AREAS)

SOFTWARE SYSTEM INTEGRATION

TARGET SYSTEM INTEGRATION

FIGURE 1.

modules are tested first, then the modules that call them are tested; and so on up the hierarchy until the software system is completely treated as an entity. However, to get the "calling" module to make the call under interesting data conditions requires "drivers" to be written for each module. There are available commercial tools to assist in doing this for certain languages. One major advantage of bottom-up testing is that it uses proven components in support of the test objective.

o Top-down testing: Although the name would imply an exact opposite definition from bottom-up testing, top-down testing usually starts at the top of the hierarchical structure, but first moves down through the levels tracing the paths that get the system's input/output working first. This is done so that the testing can then be driven by user or simulated sensor inputs. The major drawback to top-down testing is that "stubs" or simulations of missing or not-yet-developed modules are required. In very complex and/or real-time software these can represent a quite formidable and expensive undertaking.

System Testing

One obvious property that each "level" of testing should have is the ability to provide intelligent feed-back to lower levels in case of problems. This means that system level testing strategy must be more than mechanically putting a system through its paces, even if the test design and instrumentation are well thought out relative to a system level spec. The system tester must look at the system through the software's "eyes" as being a collection of hardware to be controlled. He, therefore, cannot treat software as a "black box." On the other hand, he does not have time to deal with the code-level problems. He must do "gray box" testing. His test case selection (which is usually limited by time and money) must include scenarios that exercise high problem areas or areas where real world environment may be quite different from the environment simulated in earlier tests. This means that his monitorship of earlier activities is essential. Further if he is to provide intelligent feed-back, instrumentation must be provided. This will be discussed later. There may also be a requirement for a system level driver. The system level tester should know this by comparing the system performance requirements to his capability to produce inputs at the levels required. When a driver is needed, it should be made known early enough in the program to have it available in a timely fashion. Meyers (8) discusses 14 categories of system level testing that may be required. All of these point to a general strategy of starting early, assuring that prerequisite functions (requirements/specs validation, instrumentation, drivers, simulations, etc.) are done, and designing tests that not only verify system requirements, but that also fully investigate problem areas.

TESTING METHODOLOGIES

The following discussions cover three of the more important test methodologies; requirements, simulation, and instrumentation.

Software Requirements Analysis

For software, as well as for other major components of complex systems, one of the primary activities of the tester/evaluator is to determine whether that major component meets requirements. Whereas the methodologies for this type of activity have been fairly well developed and employed for hardware, the same is not true for software. One of the basic problems with respect to software test and evaluation is the determination of what the software requirements are. A second problem, once the requirements have been identified, is whether they are testable. Experience has shown that software requirements are very often incompletely, vaguely, or qualitatively stated, and thus are not testable. Two additional factors which greatly complicate the problems of developing software are: the attitude that software is not a critical item, but is something to be "poured in" after the hardware is built; and the tendency to change the software to adjust for deficiencies of another system component.

A software requirements analysis involves assuring that software requirements are identified and tested. Steps which can be followed in such an analysis are: the identification of the requirements; the tracing of these requirements, both back to the system requirements and forward to the code; and the testing demonstrating that the requirements were met.

Requirements Identification

The initial problem for the tester/evaluator is to determine what a software requirement is. For most software-supported tactical and weapon systems, a software or data processing system requirements document exists. The problem is that in these documents the "requirements" are not clearly identified; further, information in these documents varies from a general discussion of a system requirement in some cases to a detailed flowchart of how to code a routine (not a requirement) in other cases. This lack of understanding of what the real software requirements are creates problems for the developer as well as for the tester/evaluator - the developer has no more idea of what to build than the tester/evaluator has of what to test and evaluate.

The first matter to be addressed is what constitutes a requirement and how it is to be identified from the maze of documentation. To be implementable and testable, a requirement does not stop with a general statement of what is to be done; it must also contain descriptive infcrmation that states its performance attributes (e.g., accuracy, volume, interfaces, etc.). Thus, the identification of requirements should include not only the basic requirement statement, but the descriptors or attributes of each requirement as well. Table 1 lists performance descriptors (PD) that will provide the minimum information necessary to implement a requirement, along with summary definitions of each PD.

Also included in the identification of requirements is the assessment of the adequacy with which the requirements are stated. This should give an indication of the readiness of the developer to proceed with implementation.* The presence or absence of PD information is indicative of the degree to which the necessary attributes of requirements are stated. This does not imply that the software requirements are responsive to the system requirements or that they are complete; it is an indication of the detail with which software requirements have been stated. Nor must every PD be specified for every requirement. However, the lower the level of documentation which must be reached before open PDs are specified, the more subject to interpretation (and therefore to error) the requirement becomes.

Identifying the software requirements from documentation such as the type A specification or data processing system requirements is only the first step in the requirements analysis. Next, it is necessary to determine the traceability of the requirements both from user requirements to code and from code to user requirements. The former will be discussed here. Both are equally important.

TABLE 1. PERFORMANCE DESCRIPTORS
FOR SOFTWARE REQUIREMENTS

1. Requirement: A general, but concise, statement of the system or software requirement. The details of the requirement will be presented in supporting columns. The requirements will be structured so that the system level requirement needing software support will be stated first, followed by the lower level software requirements. The software requirements will be distinguished by placing a dot (^{o}) before each requirement. Additional levels of dots (^{oo} or ^{ooo}) can be used to designate breakdown of requirements to even lower level statements.

2. Source Selection: Document from which each requirement was extracted, followed by the section number (DPSR, FS, DS, MM -3.2.1).

3. Implementing Process: The software unit which is intended to provide the logic needed to satisfy the requirement. For the system requirement, the implementing process will be the major software function(s) which provide overall control.

*The judgment of the adequacy of the requirements specification is typically developer, not tester, responsibility. The tester's concern with requirements testability is, however, apparent.

4. Input: The data (tables, files, or other form) which are necessary to fulfill the requirement. The general guideline is to provide only that information which is required external to the implementing process. Data required in local processing should not be included.

5. Processing Condition: The statement of what conditions, events, system status, etc. (i.e., the software environment), must be present before the implementing process control is provided.

6. Output: Data (tables, files, or other form) provided at the completion of the implementing process. See input for additional information.

7. Constraints: Any rules, regulations, etc., which are imposed on the software system as a whole, i.e., priority structure, etc. These are generally determined through review of the supporting information for the requirement.

8. Executing Sequence: The order in which the Implementing Process is executed relative to other Implementing Processes can be stated in boolean form; e.g., (A) (B) (C). Conditions under which control is passed should be noted. No attempt should be made to force several or all Implementing Processes into a sequence. Processes should be related as specifically stated in the DPSR or as can be determined from related information within the DPSR.

9. Error Response: A statement of what course should be followed in case of an error, such as voids in data, hardware failures, overloads, etc.

10. Processing Volume: The maximum level of processing per unit of time which the system should be capable of, i.e., five tracks, four missiles.

11. Accuracy: The accuracy to which processing should be carried; i.e., tracking of a target within certain boundaries, range to 1 foot, etc.

12. Time: Any constraints on amount of time the implementing process has to complete its function; e.g., time the process should be initiated or time the process should be completed.

13. Sizing: Resource requirements needed in response to the stated implementing process; i.e., memory, tape, I/O channels, etc.

14. Cross Reference: Any reference to any other document, section of a document, or processing unit which will provide supporting information or execution support.

15. Comments: Any additional information needed to fully describe the subject being discussed.

Requirements Tracing

As previously stated, the implementation documentation will, in many cases, provide details of the requirement itself in the form of PDs which were not detailed in the requirements document. In this sense, the implementation documentation not only describes how each requirement will be implemented but completes the description and defines additional requirements.* Thus, a first result, or by-product, of this trace is completion of the definition of software requirements. The real products of this trace are information on:

(1) Software requirements which are satisfied, totally or partially, by the implementation (including both program structure and function).

(2) Software requirements omitted in the implementation.

(3) Implemented structures or functions which are not necessary to support the software requirements.

Restated, this downward trace provides information on the completeness and adequacy of the implementation, including requirements which are missing from the implementation, and the exposure of extraneous implementation. Such conditions could reflect a misinterpretation of requirements, an expansion or redefinition of the actual requirements, the addition of unrequired "niceties", or a variety of other unauthorized occurrences which would increase the cost and time of development and decrease the quality of resultant software.

The requirements trace, then, is an analysis conducted early in the development cycle to assess the responsiveness or conformity of the software implementation to the system requirements. The alternative is to rely on test results later in the development cycle, at a point where the cost of detection and elimination of errors has substantially increased. In addition, absolutely complete testing of software for complex tactical and weapon systems is neither physically possible nor cost effective. This does not mean that an acceptable (and in that sense "complete) test program is unattainable; but it does seem to place added emphasis on the necessity for requirements tracing. Software requirements can be traced downward by comparing information from the

*This is stated here simply as a matter of fact. The common practice of partial or incomplete statement of PDs in requirements-level documents does not provide for a strong requirements baseline.

type A, type B5, and type C5 documents.

Testing to Requirements

The test design process is much more complex and involved than generally considered. It is not the specification of a few system-level tests to be conducted toward or at the end of the development process; it is a multileveled, multifaceted operation which must be considered throughout the entire development cycle. It involves a detailed knowledge and comprehension of the software and system requirements as well as the design of scenarios to test these requirements. This is of particular significance with large, complex software packages, where complete testing is impossible. The test designer must, based on his knowledge of the requirements, PDs, and implementation, identify a limited number of cenarios or test cases on which to base his evaluation. It would appear that the overall familiarity with requirements and the detailed understanding of the above, equip the analyst for test design.

This leads to another important consideration in test design - that of collecting data from each test. Clearly, the evaluation of any test is dependent not just on the information produced by a test, but on the information collected during that test. Typical of many software controlled or supported military systems, some of which are still being developed, is the inadequacy of or even disregard for information and data collection on software/computer system performance. Collection and recording of data during the operation of a real-time software system does present some problems, in particular the impact of the time required for collection/recording on the real-time process. The seemingly obvious solution lies in the philosophy that the capability for collecting and recording, and the time required for this during real-time operations should be designed into the software/computer system.

Methods or techniques for software testing are more expensive than one might anticipate, and more diverse than a narrow definition of testing might allow. They can be categorized as either static or dynamic: static methods are those which do not involve the execution of (i.e., passing of data through) the target software; dynamic methods are those which do involve execution of the target software. Thus, testing can include not only testing to design and functional requirements in the dynamic sense, but also in the static sense, such as documentation analysis and simulation.* In view of the impossibility of

*Design requirements are those which relate to the program structure or architecture. Included are timing, sequencing, and management and control of the software/computer system. Many of the executive/operating system requirements and the software control (threading) requirements fall into this class. Functional requirements, on the other hand, are those which relate to implementation of math/logic models, accuracies, volumes, and management and control of hardware (e.g., radar, missile). Most of the application software requirements fall into this category. Often it is difficult to distinguish software functional requirements from system functional requirements.

complete dynamic testing, these static methods become necessary forms of testing. This report will not discuss each method and its application to dynami- and/or functional requirements; however, it should be noted that there is more than one way to approach the problems of comprehensive software testing.

Role of the System Tester/Evaluator

The role of the tester/evaluator in requirements testing varies from unit through integration and system level testing. Initially the role in testing is one of passive involvement, shifting to a more and more active involvement as the test program progresses. At the unit test level, the tester should have a general understanding of the unit test program and philosophy so he can advise the project manager of its adequacy. This involvement by the tester, in keeping with the single integrated development test policy, is necessary for two reasons:

(1) Because of the limited amount of system level testing that can be performed, demonstrations of many of the functional requirements and capabilities of the software which occur during unit testing cannot be repeated in the higher level tests.

(2) Because of the criticality to the tester/evaluator of the data collection and reduction capability during the higher level tests, evidence of an adequate data collection and recording capability must be demonstrated during unit testing.

At the integration test level, the tester/evaluator's involvement becomes more active. Here many of the top level requirements relative to software architecture and hardware-software interfaces will be demonstrated. The tester/evaluator should assess the adequacy of the integration test program, identify additional testing needs and assist in the integration of them into the test program, participate in establishing acceptance test criteria, and participate in the evaluation of acceptance test results. At the system test level, the tester/evaluator is again very actively involved. Whereas integration level tests are oriented more towards demonstrating that subsystem (software in this case) level requirements were met, system level tests are oriented toward demonstrating that the integrated subsystems meet system requirements. The tester/evaluator should assess the adequacy of the system level test program, identify additional testing needs and assist in the integration of them into the test program, participate in establishing acceptance test criteria, and participate in the evaluation of acceptance test results.

Again, it is not sufficient for the system tester/evaluator to become involved only at system level testing. Many of the software/system requirements are not demonstrated, or even demonstratable, at this level.

Furthermore, if deficiencies are noted (e.g., inadequate testing, insufficient data collection and recording, or even incomplete or inadequate statement of requirements), it is too far into the development cycle to have impact without severe or potentially severe repercussions in schedules and costs.

Software/Computer System Simulation

The application of simulations in the test and evaluation of complex systems and their subsystems is a well-established precedent in both industry and government. Since any type of testing short of a full-up field test involves some degree of simulation, this is not surprising. All of the software testing during unit-and integration-level and probably during most of the system-level testing would qualify as some for of simulation.

Most test and evaluation techniques are addressed to the functional operations. It is of little tactical significance to have an ideal tracking filter coded into the software if the program units which contain that filter cannot be enabled and provided with the processing resources (e.g., CPU time, memory access) necessary to perform those operations; i.e., if the design of the total software process cannot support the functional operations. The system sensitivities to software design are generally not observable under nonstressing conditions; however, under stressing or high-load conditions, where processing resources are at a premium, these sensitivities are observable. Simulation of the software design provides a mechanism for testing and evaluating that design under a variety of load conditions.

Two approaches to software/computer system simulation are the use of a "package" simulator and the use of a "language" simulator.

(1) A package simulator provides a generalized algorithmic model of a software/computer system into which a user can put those parameters which define his system. A primary purpose for using a package simulator is that the model itself is prebuilt, and the period for coding, checkout, and validation of the model is avoided.

(2) A language simulator, on the other hand, is actually a specialized programming language tailored to the specification and simulation of software/computer systems.

Conceptual Level Simulation

Conceptual level simulation is done at a very high level, considering each major software function (or operation) as a single job. This type of modeling would be appropriate early in the software development process. It should be done as soon as the basic software requirements, the type A specifications, are identified, before the code is written Its basic purpose would be to provide an interactive model of the software requirements, in order to assess initial process designs, and to provide initial timing and sizing estimates. (The type of modeling and simulation suggested at this level may well be pencil-and-paper studies not requiring the use of automated or computerized simulation techniques.)

Task Level Simulation

Task level simulation breaks the software functions into their component tasks. This breakout could be at any of several levels of detail, depending on either information availability or the purpose of the simulation, or both, and would be based on the type B5 computer program development specifications and/or type C5 computer program product specification.

Software/Computer System Instrumentation

One of the most vital prerequisite functions that must be performed is the planning for and procuring of instrumentation and test tools. Many automated tools exist on the open market today. (In fact, one of the most comprehensive glossary of software tools and techniques is provided in reference 9.) Probably the least understood of the instrumentation is the use of monitors for system level testing.

A variety of both hardware monitors and software monitors is available in commercial systems. Monitoring problems arise, however, with respect to special-purpose software/computer systems such as those generally associated with military tactical and/or weapon systems.

Software Monitoring

A software monitor is usually thought of as a special routine or program incorporated into the executive software which will cause certain data related to the processing which is occurring to be collected. Typical of the data collected are routine processing sequence and processing history, routine start and stop times, executive route intervention, missed deadlines, and processor idle time.

Although not a software monitor in the purest sense, another type of data collection which is built into a software routine is the collection of data related to the functional performance of that routine.

This type of collection is appropriate for applications programs and includes target data, missile data, radar data, etc. Little analysis can be performed without these types of data during maintenance, as well as during the development phase of software acquisition.

A critical feature of data collection is that it must be designed into the software. This is particularly significant for real-time systems operating on a strict data processing budget and for highly interactive systems where the time and point of collection are critical. Tactical and weapon systems qualify on both counts. In order to assure efficiency and appropriate data collection, software monitoring capability must be considered and included in the initial software design, and the resources necessary to provide collection (CPU time, memory, etc.) must be considered when the determining the system's data processing requirements. Data collection code added as an afterthought leaves a trail of problems and has several major drawbacks: it affects the overall timing and interaction of the various software modules; it introduces untested code into previously tested code, which now must itself be retested; and it is often accompanied by the implicit assumption that it will be removed as soon as the testing it supports is over. The removal of data collection code is a very questionable practice, since modules which have been successfully tested are now being modified by the deletion of this code; this leaves serious doubt as to the validity of the residual code. The axiom "fly what you test" has been demonstrated to be appropriate for both large and small software systems. However, it is not uncommon for a developing contractor to include data collection capabilities in a software package for his own testing, only to remove part or all of them prior to delivery of that package (as part of a tactical/weapon system) to the Government, thus delivering a substantially different set of code than was proven through the test program. This results in greatly reduced visibility into a software package which must be operated and maintained. The next step is obvious: pay "someone" to incorporate a data collection capability into this just-delivered software package.

Hardware Monitoring

A hardware monitor is a probe which, when physically attached to some element of a computer system, is capable of detecting when a signal passes to or through that element. Several such monitors are commercially available. Depending on the sophistication of the monitor, 100 or more such probes can be in operation simultaneously, relying on a minicomputer to process the signals for recording on a storage device (usually magnetic tape). The number of signals to a particular probe can be counted, or in some cases, the content of the signals can be decoded. Further, the characteristics of these probes are such that they produce no distortion, noise, interference, or other side effects which could either negate the validity of the data collected by them or affect the performance of the equipment to which they are attached. Typical measurements made via hardware monitoring include: CPU active/wait time, channel activity (channel busy - CPU active, channel busy - CPU wait), peripheral activity (seek in process - CPU wait), and designated software job activity (number of times enabled, length of time active).

SUMMARY

This paper has attempted to emphasize and to some degree inform the reader on software testing. It is written through the eyes of the system tester, but not (it is hoped) to the detriment of the other testing levels. There is much literature and general knowledge about the middle ground of software testing: e.g., program structure design, module design, coding, module testing. However, for the very early aspects---requirements, objectives, external specifications, system architecture---and the very late aspects---external function testing, system testing---very little relative knowledge exists. Until such time as the body of knowledge is filled in software testing, the project manager's best policy is to include and seek the input of all the test participants from the beginning.

REFERENCES

1. Information Processing/Data Automation Implications of Air Force Command and Control Requirements in the 1980s (CCIP-85): Highlights, Vol I, SAMSO/SRS-71, April 1972.

2. Blown Balloons, Aviation Week, September 20, 1971, p. 17.

3. Liklider, J. C. R., Underestimates and Overexpectations, Computers and Automation, Vol. 18, August 1969, pp. 48-52.

4. Methodology Investigation Report, Validation of TECOM/ARMTE Software Methodology, August 1976, TECOM Project 7-CO-RDT-WS1-005.

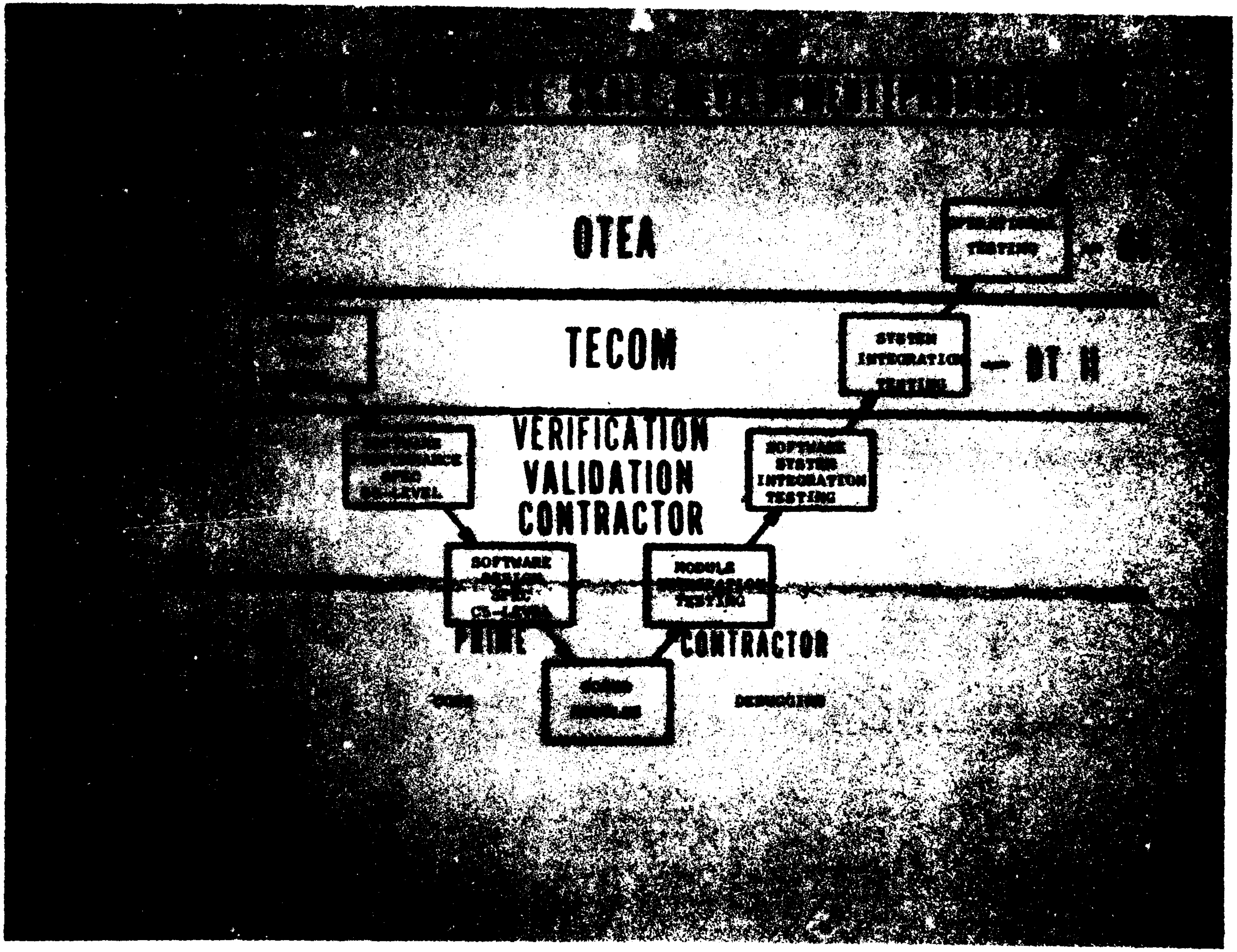
5. Gates, Howard P., Jr., et al, Electronics-X: A Study of Military Electronics with Particular Reference to Cost and Reliability, Institute for Defense Analysis, August 1973.

6. Boehm, Barry, Software Engineering, IEEE Transactions on Computers, Vol. C-25, No. 12, Dec 76, pp. 1226-1241.

7. Kossickoff, Alexander, Software Requirements Analysis Validation; Part I: Rationale and General Approach. The Conference on Software Management in Defense Systems and Other Federal Programs, 1976.

8. Meyers, Glenford J., Software Reliability Principles and Practices John Wiley and Sons, Inc., 1976.

9. Reifer, Donald J. and Trattner, Stephen, A Glossary of Software Tools and Techniques, Computer, July 1977.



OTEA

TESTING

TECOM

SYSTEM INTEGRATION TESTING

DT II

VERIFICATION
VALIDATION
CONTRACTOR

SOFTWARE SYSTEM INTEGRATION TESTING

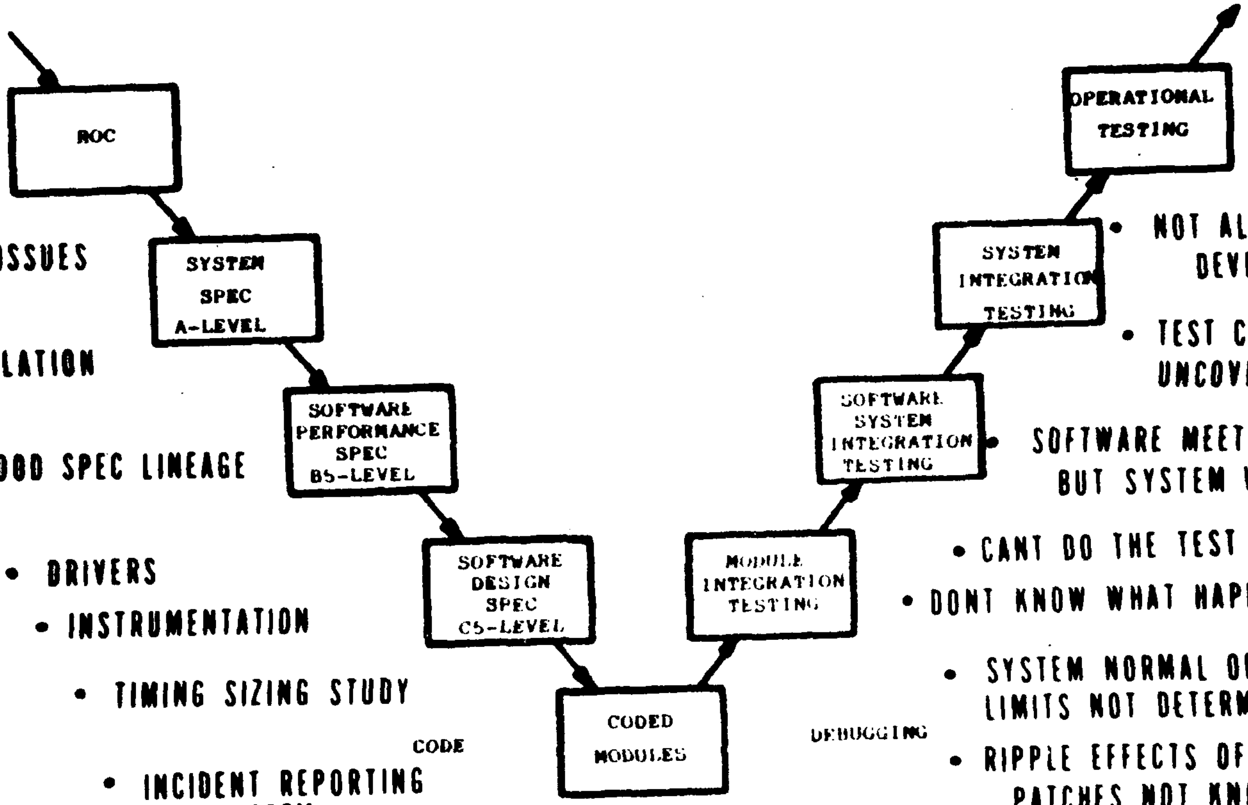
SOFTWARE DESIGN SPEC CS-LEVEL

MODULE INTEGRATION TESTING

PRIME CONTRACTOR

DEBUGGING

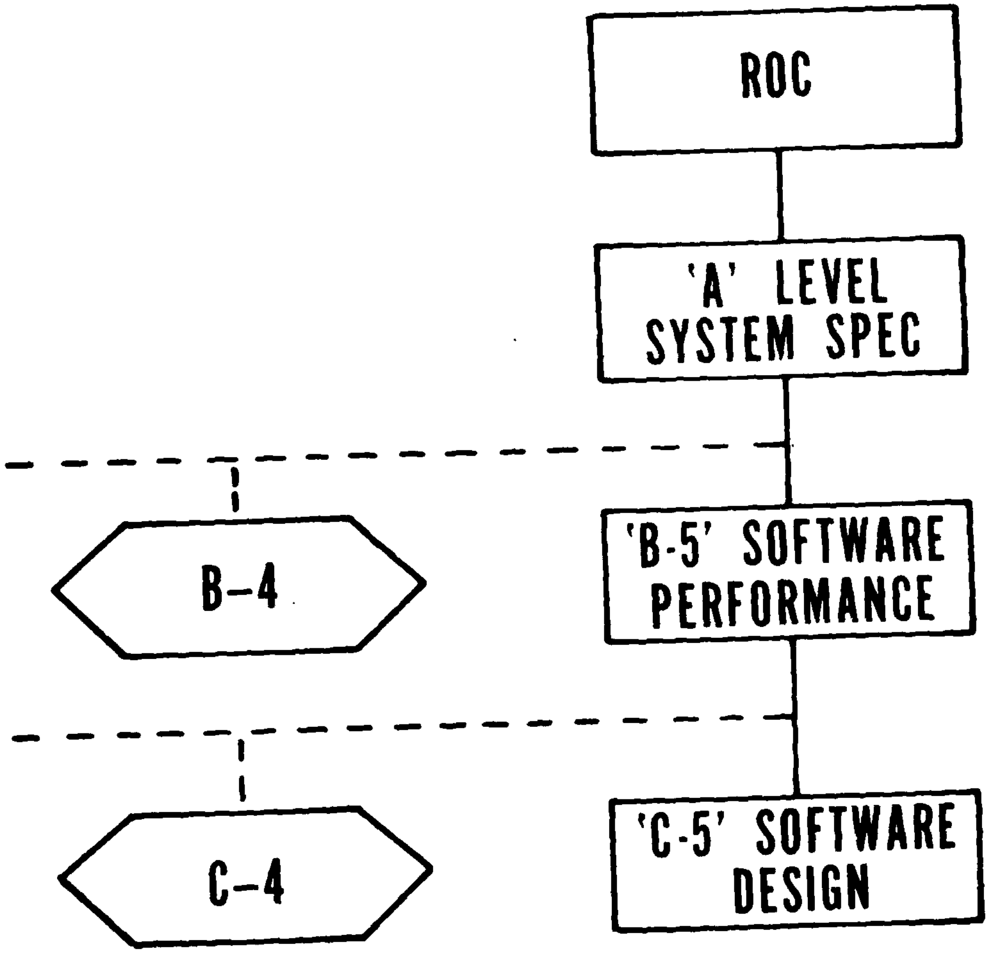
SOME EARLY ACTIVITIES FOR GOOD REASON


ROC
SYSTEM SPEC A-LEVEL
SOFTWARE PERFORMANCE SPEC B5-LEVEL
SOFTWARE DESIGN SPEC C5-LEVEL
CODE
CODED MODULES
DEBUGGING
MODULE INTEGRATION TESTING
SOFTWARE SYSTEM INTEGRATION TESTING
SYSTEM INTEGRATION TESTING
OPERATIONAL TESTING

- CRITICAL ISSUES
- SIMULATION
- GOOD SPEC LINEAGE
- DRIVERS
- INSTRUMENTATION
- TIMING SIZING STUDY
- INCIDENT REPORTING SYSTEM
- CONFIGURATION CONTROL

- NOT ALL FUNCTIONS DEVELOPED
- TEST CASES DO NOT UNCOVER PROBLEMS
- SOFTWARE MEETS B-5 SPEC BUT SYSTEM WONT WORK
- CANT DO THE TEST
- DONT KNOW WHAT HAPPENED
- SYSTEM NORMAL OPERATING LIMITS NOT DETERMINABLE
- RIPPLE EFFECTS OF SOFTWARE PATCHES NOT KNOWN
- SOFTWARE DOESNT MATCH DOCUMENTATION

1 OZ OF PREVENTION 1 LB OF CURE



ROC

OPERATIONALLY, WHAT HAS TO BE DONE ?

'A' LEVEL SYSTEM SPEC

FUNCTIONALLY, HOW WILL THE SYSTEM DO IT ?

'B-5' SOFTWARE PERFORMANCE

B–4

QUANTITATIVELY, HOW WILL THE SOFTWARE BE CONSTRUCTED AND PERFORM ?

'C-5' SOFTWARE DESIGN

C–4

DETAIL DESIGN, FLOW CHARTS, ACTUAL CODE.

AUTOMATIC TEST & DIAGNOSTICS

Milton Tenzer
CENTACS

AUTOMATIC TEST & DIAGNOSTICS

SESSION CHAIRPERSON: Milton Tenzer

SESSION SUMMARY

Software development for automatic test and diagnostics systems generally involved a two-step procedure in which the user requirements are first translated into a test specification and then this specification is extended into a computer program. The first paper described the design attributes and features of a modern higher order language (OPAL), which has recently been developed specifically for test programming. The second paper described the two-step automation of software development to test and diagnose electronic circuit boards/modules. A closely related non-operational procedure language (NOPAL) is used to analyze the requirements and to generate the test specifications. The programming can be performed in IEEE ATLAS or OPAL. The third paper examined certain technical issues relating to built-in test techniques for the evaluation of integral fault detection and isolation capabilities for modular programmable digital computers and assess their built-in test performance versus cost trade-offs.

OPAL - A MODERN LANGUAGE FOR TEST PROGRAMMING
by H. Kaunzinger, CENTACS

NOPAL: AUTOMATIC TEST PROGRAM GENERATION by N. Prywes,
Professor of Computer Sciences, University of Pennsylvania

SOME ISSUES RELATED TO BUILT-IN TEST FOR COMPUTER SYSTEMS
by J. Clary and Atul R. Jai, Research Triangle Institute

OPAL - A Modern Language for Test Programming

Helmuth M. Kaunzinger

Software Engineering Division
CENTACS

OPAL stands for Operational Performance Analysis Language and its development has been sponsored by the U.S. Army. Unlike the ATLAS language, which has grown into a test specification and test procedure description language in a large committee with little regard to programming problems, OPAL has been designed as a modern language for all phases of testing with emphasis on test programming. OPAL has many unique and new features not yet found in ATLAS. The most important and most cost-saving features are: modularity, the concept of ATE independence by stipulation of virtual test resources allowing ATE independent test procedure documentations, the notion of allocation of virtual-to-existing ATE resources, the unambiguous resource model in describing and programming a test to replace the outdated signal model concept, and the choice of flow control constructs for structured programming. All these features were treated in detail.

OPAL - A Modern Language for Test Programming

HELMUTH M. KAUNZINGER
US Army Communications Research
and Development Command
Center for Tactical Computer Sciences
Software Engineering Division
Fort Monmouth, New Jersey

OPAL stands for OPERATIONAL PERFORMANCE ANALYSIS LANGUAGE, an acronym created during the early development phases of this test language originally called CTL (Computer Test Language). Its development has been sponsored by the US Army when it was recognized that there was no official standard for test languages, especially for the programming aspect of it. The development started in the early 1970's building on the collective test engineering experience embodied in the early versions of the ATLAS language intended for ATE independent test specifications and procedure description. ATLAS evolved in a loose committee effort in an uncontrollable fashion.

Before going into language details, it is necessary to list the Army quality criteria for a test language. They have been established in view of the expected multimillion expenditures on test programs and their maintenance problems. Only the best language option can produce all possible cost savings the taxpayer is entitled to. These quality criteria are shown in condensed form in Table 1.

In the following, the items in Table 1 are considered from the viewpoint of cost savings. Obviously, a changing language specification leads to horrendous maintenance problems. The extra cost is directly proportional to the number of non-compatible versions to be maintained. Even compatible versions increase the maintenance cost beyond a reasonable degree. The specification of the OPAL language is so designed that functions common to all testing remain stable and are located in the core of the definition. Yet on the periphery of the language definition, there is room for low level nouns-, modifier-, and unit-expansions to keep the language flexible for expression of novel concepts evolving with the rapidly growing test technology.

Tracking of program with procedure are the prerequisite for inter-ATE-transferability. Expressed in terms of cost efficiency, this transferability is the quotient of the ATE independent procedure statements over the total program (being the sum of approximately 95% procedural statements and approximately 5% ATE dependent resource allocation statements). Inter-ATE-transferability of programs has been neglected by many system planners, however, they should be reminded that they may be forced to support a 20-year life-cycle system with a five-year life-cycle ATE constructed with commercial grade components. If a change in ATE support is needed, a 95% inter-ATE-transferability may come in handy.

Table 1. Quality Criteria for a Test Oriented Language

Stability of Language Specification -(With built-in flexibility to express novel concepts from evolving technology without need to change spec.)

Tracking of Test Program with Procedure Documentation - without translation

Inter-ATE-Transferability - by ATE-independent concept of requiring programmer defined virtual resource description and specification

Resource Model Orientation - eliminates ambiguities associated with signal Model used in older test procedure descriptions

Modularity - offers substantial life cycle cost savings by binding existing validated and proven compiled modules before execution

Readability - of maximally English-like statements provides direct documentation access to all echelons involved in procurement, design and maintenance of the supported system

Modern Flow Control - provides baseline for orderly structured programs

Orderly Hierarchy - needed when OPAL is the target of automatic test program generation.

Modularity of test programs is the most cost saving attribute of OPAL. When a module is successfully compiled and validated, it remains available in mass storage for subsequent use with other modules describing other test actions. Although there are no current statistic data on savings through test oriented program modules available, it is estimated that modularity alone will result in approximately 20% cost savings in terms of mass storage space savings and, above all, program procurement and maintenance cost.

Readability saves substantial expenditures of training of all personnel required for program and documentation reviews (not test programmers) considering a reasonable personnel turnover in all phases of the system life-cycle.

Modern flow control combined with an orderly hierarchy in language constructs do not only save cost in programmer generated programs, they also make OPAL a highly cost efficient target of automatic test program generation software.

After several iterations in design and an extended maturing process using the latest techniques on language design as well as on expressability of the growing testing technology, OPAL has finally attained stability and the status of a cost-effective test language standard. In the following sections it is attempted to give a very condensed description of this language.

The document defining and specifying the OPAL language (MIL-STD-1462A, 31 Mar 78) has almost 500 pages. The syntax is described in Extended Backus Naur Form (EBNF) and might fit on 100 pages. However, it was the full intent of the language designers to inhibit liberal implementations in various compilers or interpreters. Such liberal interpretations of the language would lead to an implied proliferation which is certainly not intended by the Army. Therefore, the syntax has been supplemented by highly detailed Semantics and Constraints (approximately 70% of the text). Approximately 10% of the document has been devoted to examples.

For the frame of this presentation it was attempted to consolidate all major features of OPAL on three figures containing syntax. It was attempted in the three-figure language summary to preserve the EBNF notation used in the governing specification MIL-STD-1462A to the largest extent possible. However, space limitations forced an approach showing branches graphically with arrows rather than in the usual separate repetitious breakdown into increasingly detailed productions. Maintained were the standard parentheses holding arguments or subordinate values, the standard curly braces defining mandatory options, the standard vertical bar showing alternate options, the standard square brackets showing optional options, and the standard sequence of three dots meaning a construct repetitious of zero to many times.

Lines ending either in arrow heads or in open double corner brackets show the logical breakdown either into a lower level construct, or into the begin of an explanation (the end of which is shown in closing double corner brackets). After many trials to condense the essential language parts into three pages, this approach appeared to be the most acceptable solution.

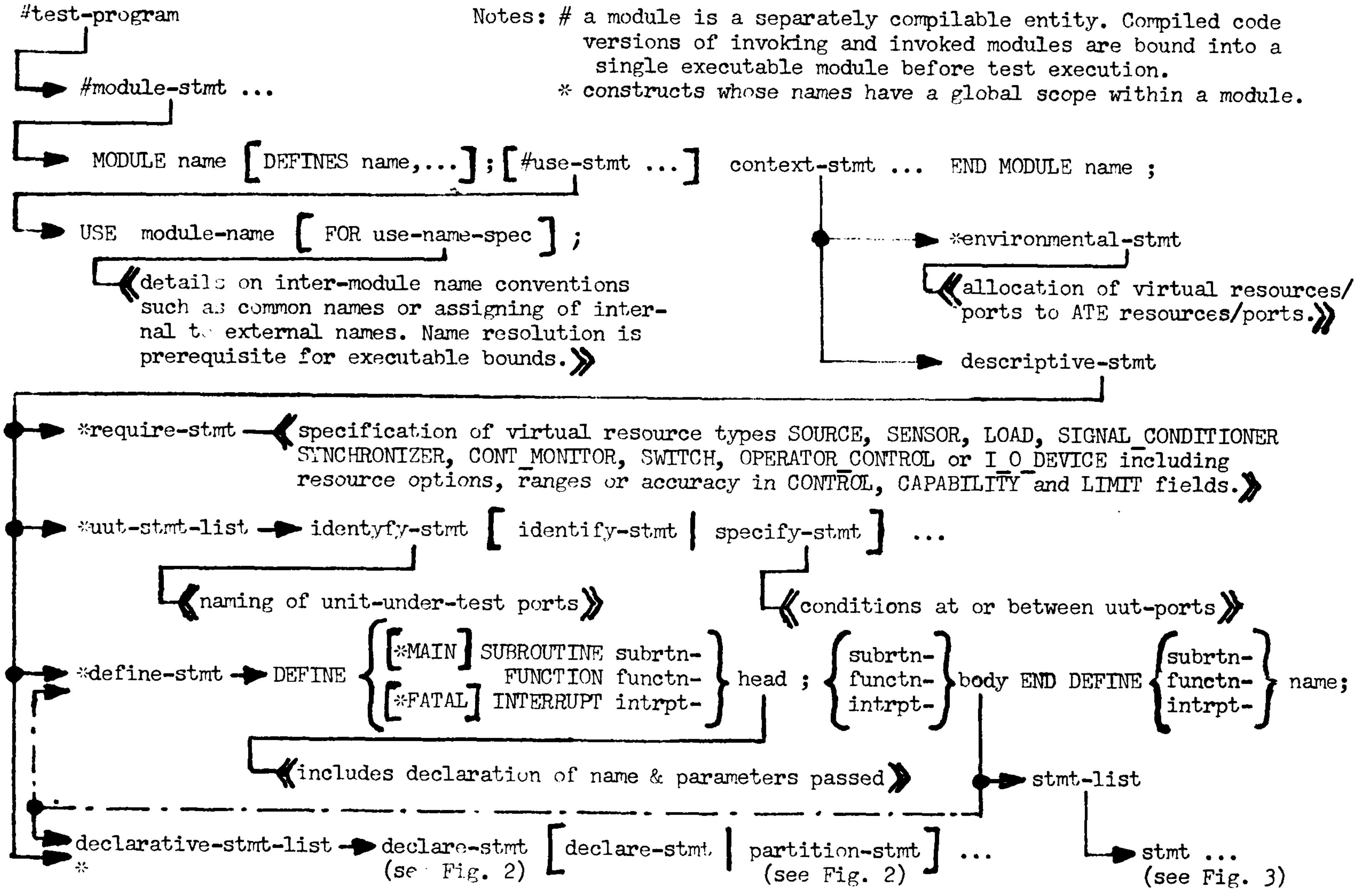
Starting in Figure 1, a test program consists of one or several separately compilable modules. Each module has an optional limit on names usable outside its bounds in the DEFINES clause. When omitted, all names global in this module are exportable to other modules. One or more optional use statements permit invocation of existing modules to become part of the executable code. This compiled code of several modules is bound into a single execution module immediately before execution by the binder software performing the usual tasks of a link editor and specific tasks involving test resource and uut related data matching at binding time. If there is only a single module without invocation of other modules, it is directly executable.

The only mandatory elements of any module are a number of context statements. Approximately 5% of these are environmental statements governing the allocation of programmers defined virtual resources and their connection parts to existing ATE resources and their corresponding connection parts. The remaining bulk of 95% consists of descriptive statements to specify the chosen test procedure.

The descriptive statements are subdivided into four very specific compound definitions:

1) Require-statements describe the needed resources to perform all tests in a given module. The details of the description cover one of nine resource types, their name(s), their controllable features, their capabilities, their limits, their tolerances and accuracies and finally meaningful names of their connection parts. The following is a description of an ac voltmeter.

```
REQUIRE DYM AS AC VOLTAGE_TRMS SENSOR WITH
   CONTROL
     VOLTAGE_TRMS=OV TO 250V WITHIN 1% PLUS 10MV
  CAPABILITY
     FREQ = 1 KHZ TO 5 MHZ
   PORTS
     HI, LO, GND;
```



Notes: # a module is a separately compilable entity. Compiled code versions of invoking and invoked modules are bound into a single executable module before test execution.
* constructs whose names have a global scope within a module.

#test-program
#module-stmt ...
MODULE name [DEFINES name,...] ; [#use-stmt ...] context-stmt ... END MODULE name ;
USE module-name [FOR use-name-spec] ;
《details on inter-module name conventions such as common names or assigning of internal to external names. Name resolution is prerequisite for executable bounds.》
*environmental-stmt
《allocation of virtual resources/ports to ATE resources/ports.》
descriptive-stmt
*require-stmt
《specification of virtual resource types SOURCE, SENSOR, LOAD, SIGNAL_CONDITIONER SYNCHRONIZER, CONT_MONITOR, SWITCH, OPERATOR_CONTROL or I_O_DEVICE including resource options, ranges or accuracy in CONTROL, CAPABILITY and LIMIT fields.》
*uut-stmt-list
identyfy-stmt [identify-stmt | specify-stmt] ...
《naming of unit-under-test ports》
《conditions at or between uut-ports》
*define-stmt
DEFINE {[*MAIN] SUBROUTINE subrtn- FUNCTION functn- [*FATAL] INTERRUPT intrpt-} head ; {subrtn- functn- intrpt-} body END DEFINE {subrtn- functn- intrpt-} name;
《includes declaration of name & parameters passed》
stmt-list
stmt ...
(see Fig. 3)
declarative-stmt-list
*
declare-stmt
(see Fig. 2)
[declare-stmt | partition-stmt] ...
(see Fig. 2)

FIG. 1 HIERARCHY OF A COST-EFFECTIVE MODULAR ENGLISH-LIKE PROGRAMMING LANGUAGE (PART 1)

2) The descriptive elements for the unit-under-test are contained in the uut-stmt-list consisting of an initial identify statement followed by zero or several optional identify or specify statements. An identify statement is confined to meaningful names for uut ports followed by an optional description of the type of connections. The specify statement may be used to state limiting physical quantities for one or any combination of previously defined uut ports. The following example shows both statements.

```
IDENTIFY X, Y, Z AS UUT STIMULUS CONNECTION;
SPECIFY FOR X, Y REFERENCED TO Z
   LIMITATION ON DC VOLTAGE_RANGE =-2V TO 4V;
```

3) Another essential descriptive statement is the define-stmt. It permits the full description procedures such as (main) subroutine, function and (fatal) interrupts in terms of a head containing naming, list of parameters and their declarations and in terms of a body leading to local declarations and action statements.

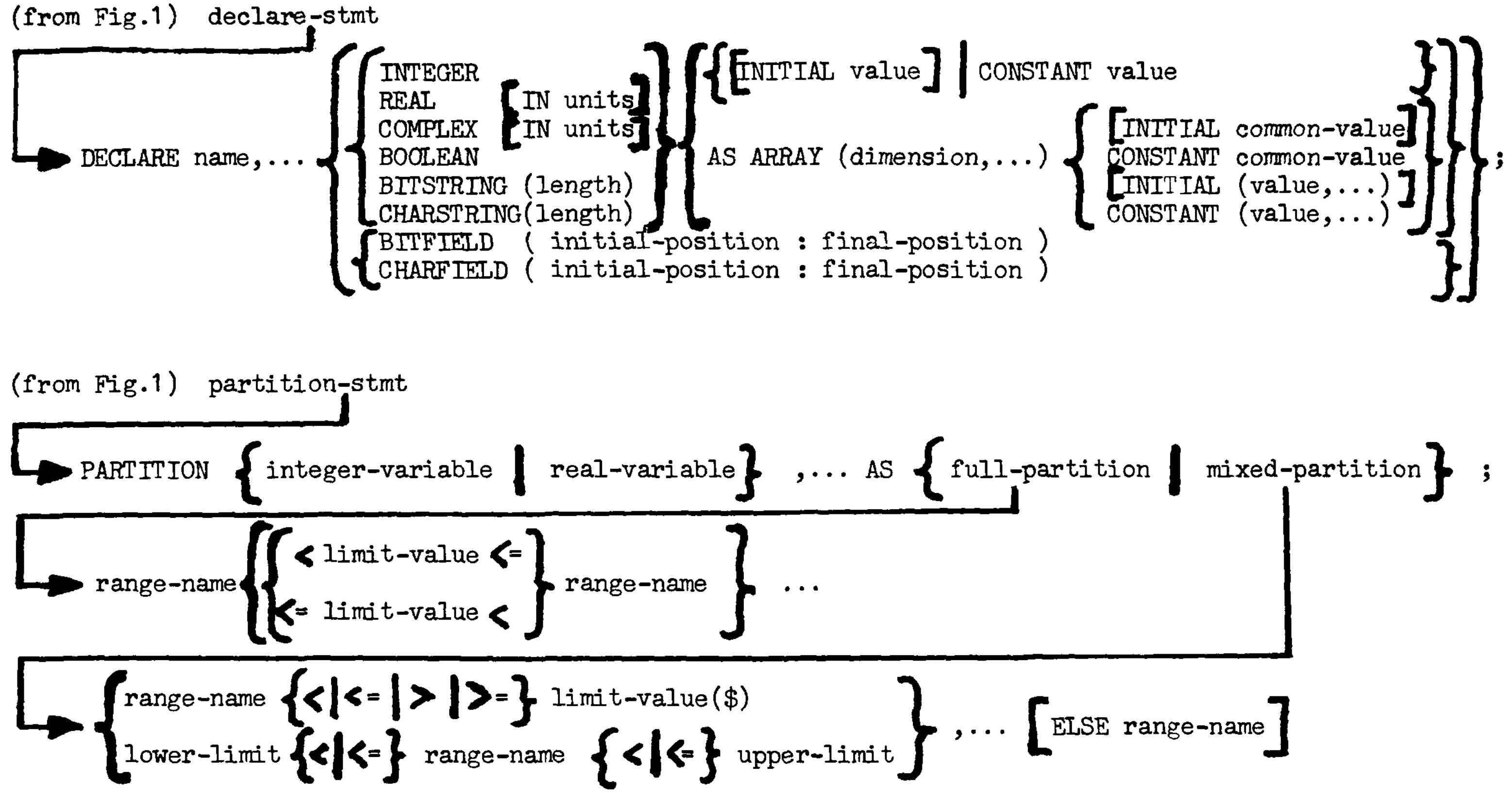
4) The last descriptive statement is the declarative statement list consisting of at least one declare statement followed by zero to several optional declare statements or position statement.

The asterisks shown with environmental statement, require statement, uut statement list, define statement and declarative statement list indicate the global naming scope of these statements. Only the define statement (exception: main subroutine or fatal interrupt) and the declarative statement list can be vested into subordinate procedure definitions.

Details on the declare and partition statements are contained in Fig. 2. It is necessary to elaborate on these two statements as the declare statement definition shows all the available data types in OPAL as well as installation of variable or constant scalars or arrays. (Existing test programming languages with data structures limited to real and bitstring types have turned out to be wasteful in core utilization and processing speed.) The declaration of separate names for bitstring fields or character fields is a powerful tool for manipulation of test patterns.

The partition statement, also shown in detail in Fig. 2 shows a novel feature in OPAL permitting declared integer or real variables to be partitioned into full (-infinity to + infinity) or into mixed ranges. The chosen range names need only be unique for each one of the variables partitioned. The following example shows a declaration and a partition of two temperature variables:

```
DECLARE TEMPIN, TEMPOUT REAL IN DEGF;
PARTITION TEMPIN, TEMPOUT AS COLD<= 50.0<COOL<=
                             65.0<WARM < 80.1<=HOT;
```



```
(from Fig.1) declare-stmt

DECLARE name,... { { INTEGER                         } { { [INITIAL value] | CONSTANT value                 } } ;
                   { REAL        [IN units]          } {                                                    }
                   { COMPLEX     [IN units]          } { AS ARRAY (dimension,...) { [INITIAL common-value]  } }
                   { BOOLEAN                         } {                          { CONSTANT common-value   } }
                   { BITSTRING (length)              } {                          { [INITIAL (value,...)]   } }
                   { CHARSTRING(length)              } {                          { CONSTANT (value,...)    } }
                   { BITFIELD  ( initial-position : final-position )                                       }
                   { CHARFIELD ( initial-position : final-position )                                       }

(from Fig.1) partition-stmt

PARTITION {integer-variable | real-variable} ,... AS {full-partition | mixed-partition} ;

full-partition:
range-name { { < limit-value <=  } range-name } ...
           { { <= limit-value <  }            }

mixed-partition:
{ range-name {< | <= | > | >=} limit-value($)                 } ,... [ELSE range-name]
{ lower-limit {< | <=} range-name {< | <=} upper-limit        }
```

Notes: In the declare-stmt the optional initial values or required constant values must track with the corresponding names in data type and array dimension when array. E.g. a name not declared as array may only be assigned a single value via INITIAL or via default specified for each data type. For array variables, either individual elements may be initialized or a common value for all elements is assigned via INITIAL or by default.
In the partition-stmt range-names are only unique for the partitioned variable.
($) In integer-variable partitions the relational operators = and ≠ are also permissible.

FIG. 2 HIERARCHY OF A COST-EFFECTIVE MODULAR ENGLISH-LIKE TEST PROGRAMMING LANGUAGE (PART 2)

When range names are used in conjunction with their variables, simple boolean expressions result and can be easily resolved in decision branches with minimal programming.

Figure 3 is also a continuation of Figure 1 and picks up the definition of "stmt" meaning any executable statement. The four major categories are control statement, input-output statement, single action test statement and multiple action test statement.

The control statements have been carefully chosen from a number of modern languages for programming scientific and data processing computers. They are, except for the assignment statement and table statement, proven flow control statements. In the context of a test programming language they provide the programmer with optimally structured programming facilities.

The table statement provides the decision table capability favored by the automotive testing community for direct decisions in single statements or indirect decisions calling on actions defined within the compound table statement structure.

The assignment statement essentially the same as in FORTRAN, ALGOL, PL/I or PASCAL and provides OPAL with an ample capability of data manipulation including an ample set of internal functions.

The input-output-statement group has been somewhat neglected in Fig. 3. It is noteworthy, however, that the OPAL programmer defines virtual resources for input-statements and output statements. Normal actions associated with real peripherals such as input or output mode of a teletype console, paging of a printer, rewinding of a tape, finding a subdirectory on a disk, etc are automatically addressed in the novel syntax of the open input/output statement and close-input/output statement addressing ATE peripheral independent actions in so specified virtual devices.

The most relevant statement category is the single action test statement group as it provides specific action verbs for each major test action:

a) Setting a resource is controlled by the first three verbs SETUP, RESET or CHANGE.

b) Defining the connection path is controlled by two verbs CONNECT and DISCONNECT.

c) Activating a resource is controlled by the verbs CLOSE and OPEN.

d) Timing of a resource (such as initiation of triggering) is controlled by the verbs START, STOP and WAIT.

e) Sensing a physical quantity by a resource and storing its value into a memory location is controlled by the SENSE verb.

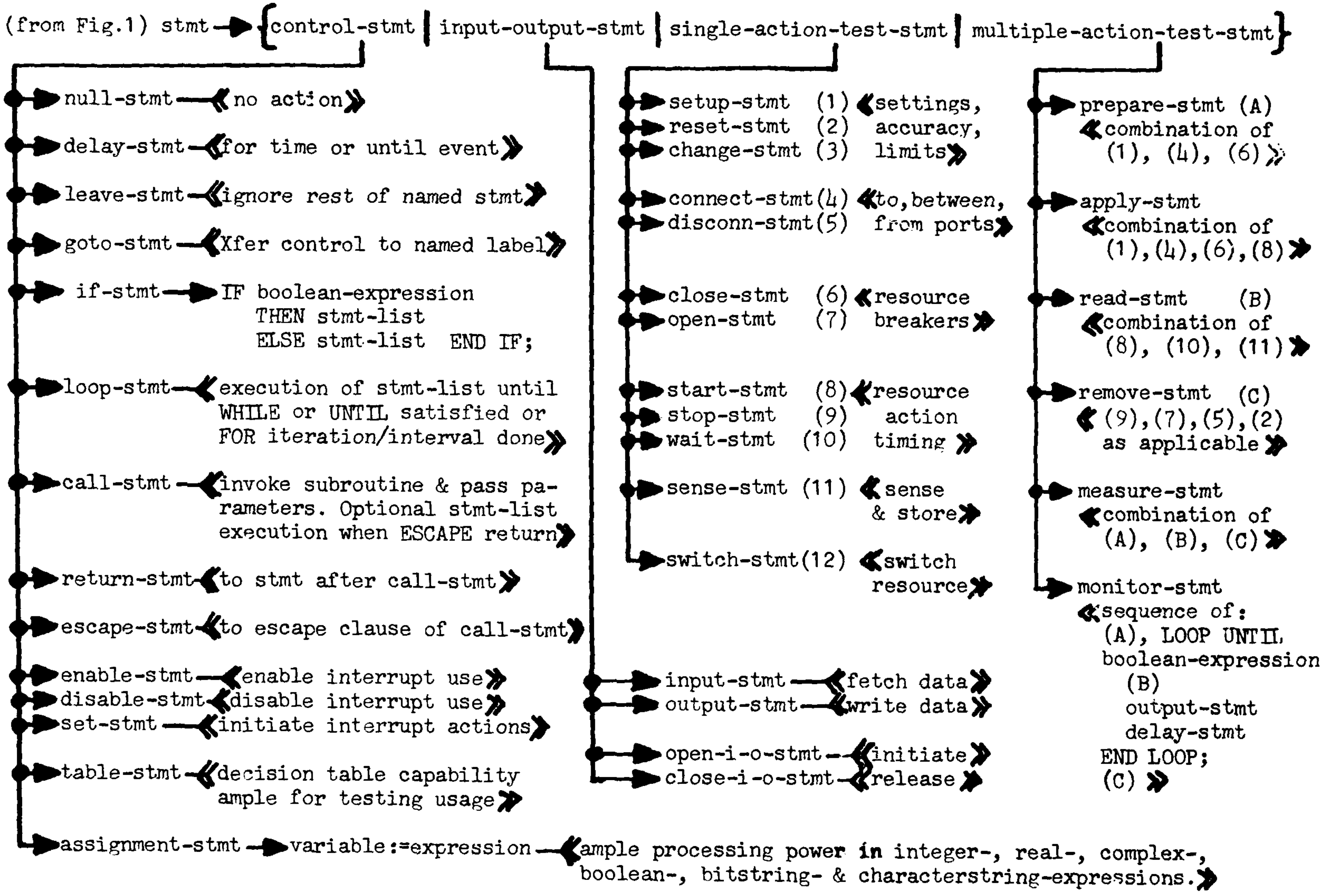

(from Fig.1) stmt → {control-stmt | input-output-stmt | single-action-test-stmt | multiple-action-test-stmt}
null-stmt — «no action»
delay-stmt — «for time or until event»
leave-stmt — «ignore rest of named stmt»
goto-stmt — «Xfer control to named label»
if-stmt → IF boolean-expression THEN stmt-list ELSE stmt-list END IF;
loop-stmt — «execution of stmt-list until WHILE or UNTIL satisfied or FOR iteration/interval done»
call-stmt — «invoke subroutine & pass parameters. Optional stmt-list execution when ESCAPE return»
return-stmt — «to stmt after call-stmt»
escape-stmt — «to escape clause of call-stmt»
enable-stmt — «enable interrupt use»
disable-stmt — «disable interrupt use»
set-stmt — «initiate interrupt actions»
table-stmt — «decision table capability ample for testing usage»
assignment-stmt → variable:=expression — «ample processing power in integer-, real-, complex-, boolean-, bitstring- & characterstring-expressions.»
setup-stmt (1)
reset-stmt (2)
change-stmt (3)
«settings, accuracy, limits»
connect-stmt(4)
disconn-stmt(5)
«to, between, from ports»
close-stmt (6)
open-stmt (7)
«resource breakers»
start-stmt (8)
stop-stmt (9)
wait-stmt (10)
«resource action timing»
sense-stmt (11) «sense & store»
switch-stmt(12) «switch resource»
input-stmt — «fetch data»
output-stmt — «write data»
open-i-o-stmt — «initiate»
close-i-o-stmt — «release»
prepare-stmt (A) «combination of (1), (4), (6)»
apply-stmt «combination of (1),(4),(6),(8)»
read-stmt (B) «combination of (8), (10), (11)»
remove-stmt (C) « (9),(7),(5),(2) as applicable»
measure-stmt «combination of (A), (B), (C)»
monitor-stmt «sequence of: (A), LOOP UNTIL boolean-expression (B) output-stmt delay-stmt END LOOP; (C)»

FIG. 3 HIERARCHY OF A COST-EFFECTIVE MODULAR ENGLISH-LIKE TEST PROGRAMMING LANGUAGE (PART 3)

f) The SWITCH verb controls only a SWITCH resource. Any test action, analog from dc to many Gigahurtz, or digital with inputs, outputs and word rate limited only to target ATE families is expressable in these single action test statements. Unlike the ATLAS language complex, there is no need in OPAL to use special purpose constructs for digital testing. This is extremely beneficial for automatic program generation.

As in ATLAS, there are multiple action test statements shown as the rightmost category in Fig. 3. The PREPARE, APPLY, READ and REMOVE verbs are semantically equivalent to the single action verbs shown in Fig. 3. The MEASURE verb is semantically equivalent to PREPARE, READ and REMOVE while the MONITOR verb is even more complex. Its semantics are shown in Fig. 3.

Single or multiple test action statements have a much similar and often repetitive structure. Consider the example:

```
MEASURE AC VOLTAGE IN V INTO VB
    USING DVM AT HI=P2, LO=P3;
```

This statement starts with an action verb, "MEASURE", followed by a noun "AC" and a modifier "VOLTAGE". The optional units "IN V" precede the storage variable "INTO VB". The next line covers the programmer-named virtual resource "USING DXM" followed by the connections between the SENSOR resource and the UUT in terms of "HI=P2, LO-P3". This multiple action test statement could have been expressed in the following single action test statements:

```
SETUP DVM WITH VOLTAGE =15 V;

CONNECT DVM AT HI=P2, LO=P3;

CLOSE DVM;

START DVM;

WAIT FOR DVM;

SENSE AC VOLTAGE IN V INTO VB USING DVM;

REMOVE DVM;
```

An important function performed by a correctly implemented OPAL compiler is to check the states of all used test resources. These states are shown in Fig. 4 in conjunction with single-action test statements, and in Fig. 5 in conjunction with multiple action test statements. Errors will occur when the prior and subsequent states of an action-verb do not match with these tables.

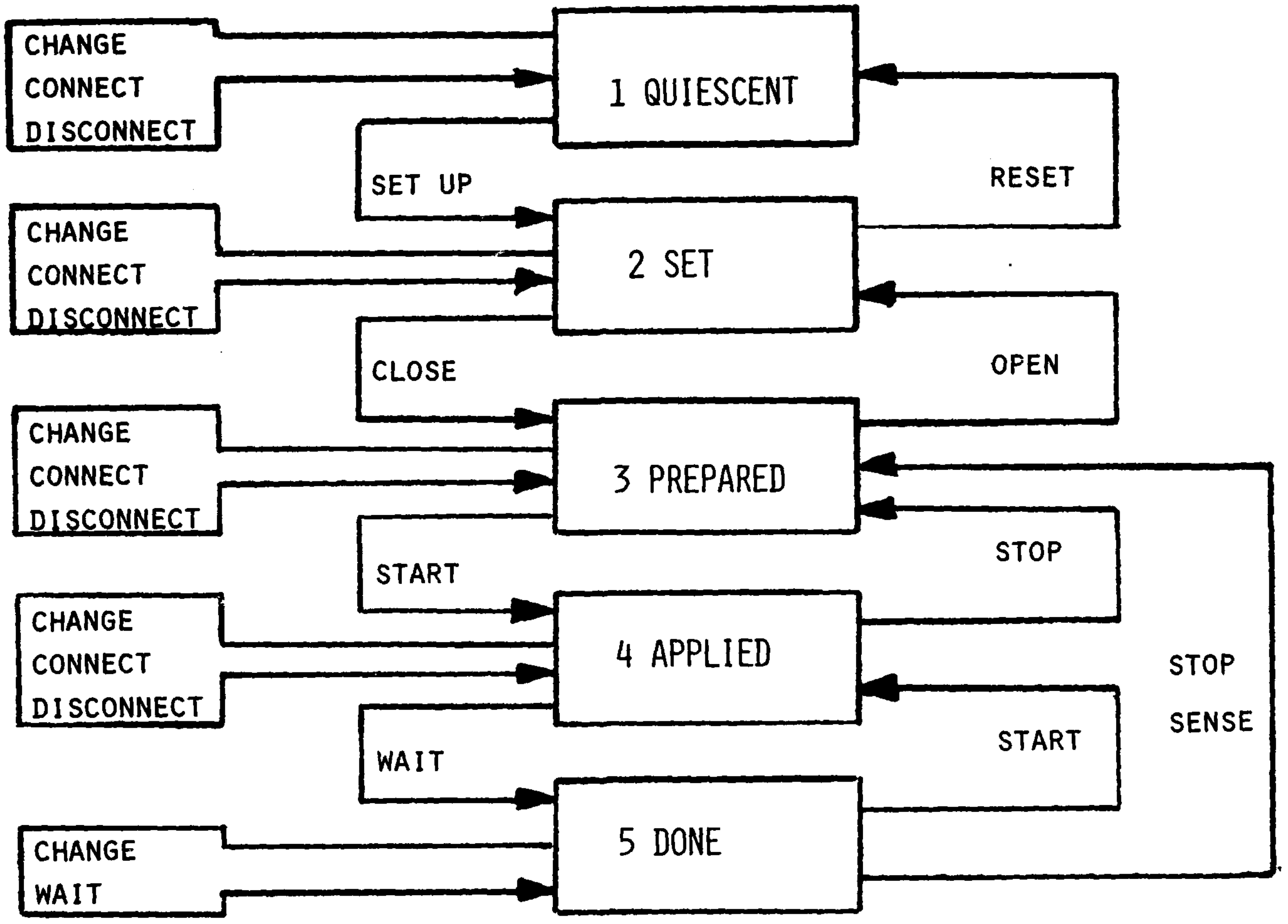


FIG. 4. STATE DIAGRAM FOR SINGLE ACTION TEST VERBS

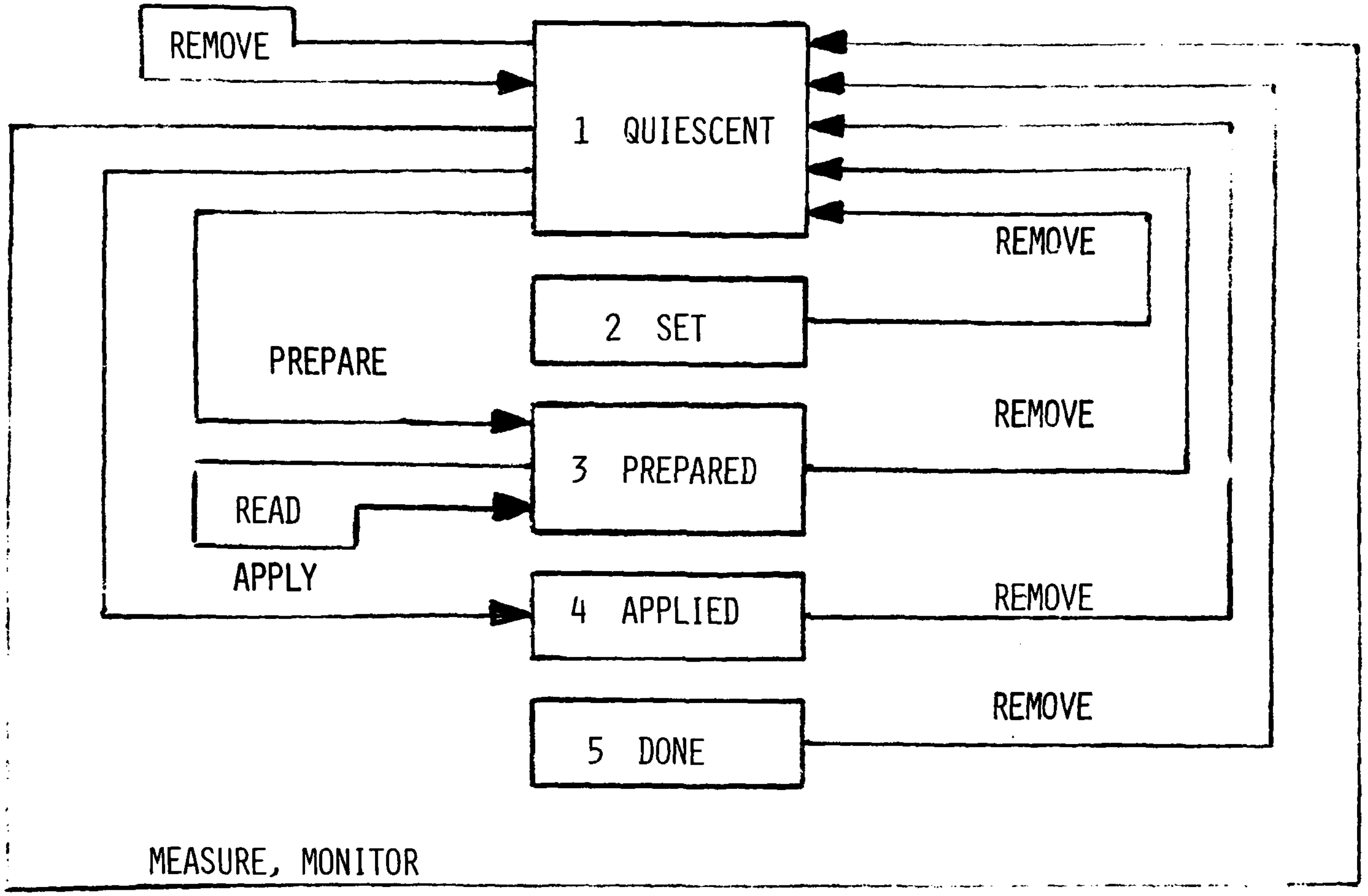


FIG. 5. STATE DIAGRAM FOR MULTIPLE ACTION TEST VERBS

Within the frame of this highly condensed description of the OPAL language, an oversimplified example of a test case with the corresponding test program is given in Fig. 6.

In the concluding section of this presentation, it is shown that OPAL fills all the needs of the chronological interaction between systems related support requirements and the corresponding ATE software support. The initial three life cycle phases: design specification, testability study, and system development, need only prudent planning and continued coordination of many aspects of the subsequent coding in the test language.

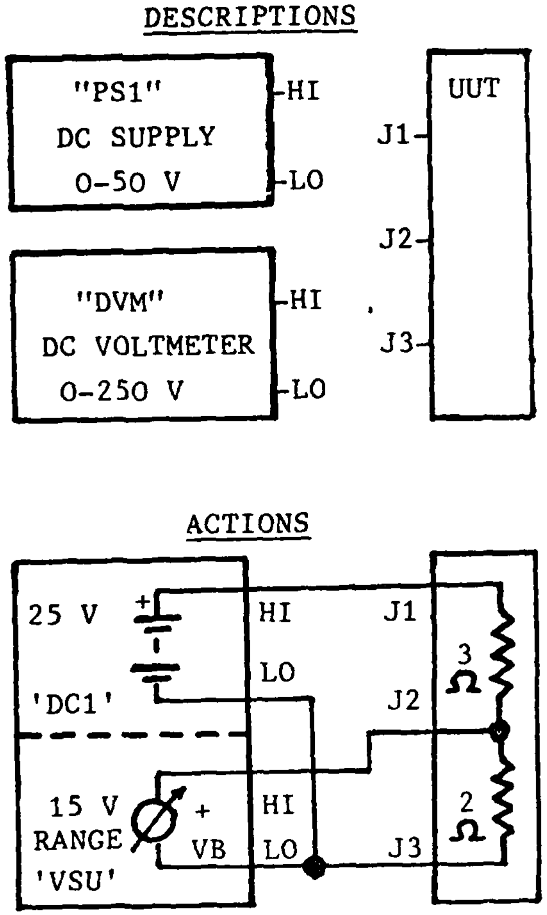
The next phase, the test specification, is the first system requirement expressable in OPAL statements. The US Air Force and the NATO Nations (to avoid ambiguities resulting from interpretation of natural languages) already insist on the use of a test language. Based on prior use they specify ATLAS. If either OPAL or ATLAS is used, translation into subsequent test-procedure descriptions cannot be avoided.

The next phase, the selected test procedure, is well documentable in OPAL. With the concept of naming and specifying ATE independent virtual test resources, this test procedure documentation in OPAL is transferable to any ATE having the described capability. With initial consideration of the intersection of all resources on several target ATE, a truly transferable test procedure will be the end product.

For the test resource allocation phase, few statements governing the allocation of the virtual resources to existing ATE resources are needed (in comparison to the number of statements for test procedure documentation) to link the ATE independent test procedure with a particular ATE architecture. The latest concept is to generate these statements automatically by resource data table matching.

The next phase, the OPAL test program is the sum of the procedural and environmental (allocation) statements of the procedure documentation and resource allocation phases of the test program development process. Although the current planning is generally based on manual programming efforts, automatic generation of test procedures and even test programs has been demonstrated in many successful pilot efforts in analog tests, and particularly in digital tests and fault analyses. However, the demands on the structural quality of a language imposed by cost-effective automatic program generation is much higher than those imposed by programmers.

A life cycle phase common to all supported systems, the OPAL language implementation, is currently scheduled for several ATE architectures now in the Army inventory (such as AN/USM-410 (EQUATE) and MATE 19). The resulting processing speed and ATE software maintainability depends largely on the quality of these compiler and run-time software development efforts.



```
MODULE DIVIDERTEST;
    USE DATAOUT;    /* MODULE DEFINING "CONSOLE"
    REQUIRE PS1 AS DC VOLTAGE SOURCE WITH
        CONTROL VOLTAGE = 0V TO 50V WITHIN 5%
        CAPABILITY CURRENT <= 10A,
                   RIPPLE_VOLTAGE <.1V
        PORTS HI, LO;
    REQUIRE DVM AS DC VOLTAGE SENSOR WITH
        CONTROL VOLTAGE = 0V TO 250V WITHIN 1%
        PORTS HI, LO;

    ALLOCATE DEVICE PS1 TO 'DC1',    /* FOR THIS
             DEVICE DVM TO 'VSU';    /* ATE ONLY

    IDENTIFY J1, J2, J3  AS UUT CONNECTION;
    DECLARE VB REAL IN V;

    DEFINE MAIN SUBROUTINE DIVTEST;
        APPLY PS1 WITH VOLTAGE = 25V
              AT HI = J1, LO = J3;
        MEASURE DC VOLTAGE IN V INTO VB USING
              DVM WITH VOLTAGE <= 15V
              AT HI = J2, LO = J3;
        OUTPUT TO CONSOLE FROM VB;
        REMOVE PS1;
    END DEFINE DIVTEST;

END MODULE DIVIDERTEST;
/* INTER-ATE TRANSFERABILITY = 833/839 = 99.3%
```

FIG. 6 EXAMPLE OF A SIMPLE OPAL PROGRAM

The next phase for a specific system, the validation of test programs, as well as the common validation of the ATE system software, may be concurrent (which usually is an oversimplification). The same comments hold for Unit-Under-Test (UUT) and ATE software maintenance.

The advances in test technology and their expressability in a stable high-order test language standard is a prime concern regarding future activities. OPAL has all the quality criteria required for such activities is also the best candidate for the subsequent integration of the newly expressable concepts from this technology evolution.

Automatic Generation of Test Programs in ATLAS*

Noah S. Prywes
Professor of Computer Science

University of Pennsylvania

Software development is generally characterized as a two-step design procedure whereby first a user's requirement statement is evolved into a specification of what is needed, and then the specification is further expanded into a computer program that satisfies the requirement. Performing this two-step procedure automatically, in addition to saving cost and development time, also provides direct interaction with the user while the automatic system verifies the consistency, completeness and non-ambiguity of the requirement and/or the specification.

The paper discussed in particular the automation of software development for computer-controlled automatic test equipment to test and diagnose the operability of electronic circuit boards. The NOPAL system that automates this software development process illustrates automation of software development in other areas as well.

Two major components of the system, referred to as the Top-Part and the Bottom-Part, correspond to the above two steps. The analysis of the requirement, the interaction with the user and the specification of needed tests, are performed by the Top-Part. The user's requirements statement must consist of a description of the circuit board that needs to be tested and the objectives of testing (whether just to determine operability or also to diagnose cause of failure). The Top-Part produces a specification of needed tests and a variety of documentation. The program design and optimization are performed by the Bottom-Part and a program in the RCA EQUATE ATLAS test language is produced.

The paper concluded with a discussion of the facilities in NOPAL to obtain wide use of it, in the automatic testing area where there is a lack of standardization of test hardware and software.

*Work performed with support from Automatic Test Systems Support, U.S. Army, Fort Monmouth, N.J.

AUTOMATIC GENERATION OF TEST PROGRAMS IN ATLAS

N. S. PRYWES

Department of Computer and Information Science

The Moore School of Electrical Engineering

University of Pennsylvania

Philadelphia, Pennsylvania 19174

1. Introduction

The subject of the paper is the automation of development of software for computer controlled automatic test equipment to test and diagnose the operability of electronic circuit boards. The paper also contains a report on the NOPAL System that automates this process.

Software development is generally characterized as a three phase procedure consisting of (1) development of requirements, (2) development of a specification for each program unit, and finally (3) development of each program in a high level language. A similar approach is generally used in automatic generation of computer programs. The NOPAL System consists of two parts: (1) a top-part which accepts as input a statement of problem requirements and produces as an output a program specification, and (2) a bottom-part which accepts as input the program specification and produces a program in the ATLAS Test Language.

Figure 1 illustrates these two parts in the context of the role of testing in the life cycle of a device or an assembly referred to in the following as a unit-under test (UUT). The UUT life cycle (on the left of Fig. 1) consists of three phases: design, fabrication and maintenance.

Information consisting of circuit diagrams, circuit layout and testing objectives is evaluated in the design phase. Changes in the design are required when testing does not satisfy the requirements. If the design is satisfactory, the top-part of the system, on the right of Fig. 1, determines a complete set of functional and fault isolation tests to be employed in fabrication and maintenance. The bottom-part produces a corresponding ATLAS program that would be utilized in computer controlled automatic test equipment (ATE), which will test the UUT and produce appropriate diagnoses. Failure statistics derived from tests are used to evaluate the UUT and modify it's design, fabrication and maintenance. Each of these parts is independent of the other and could be usefully employed by itself. The interface between the top and bottom parts is a specification of tests expressed in a language named NOPAL. Unlike ATLAS, NOPAL is not a programming language. It is non-procedural in the sense that it does not have facilities for stating commands or for sequencing the execution of tests. Tests can also be specified in NOPAL manually.

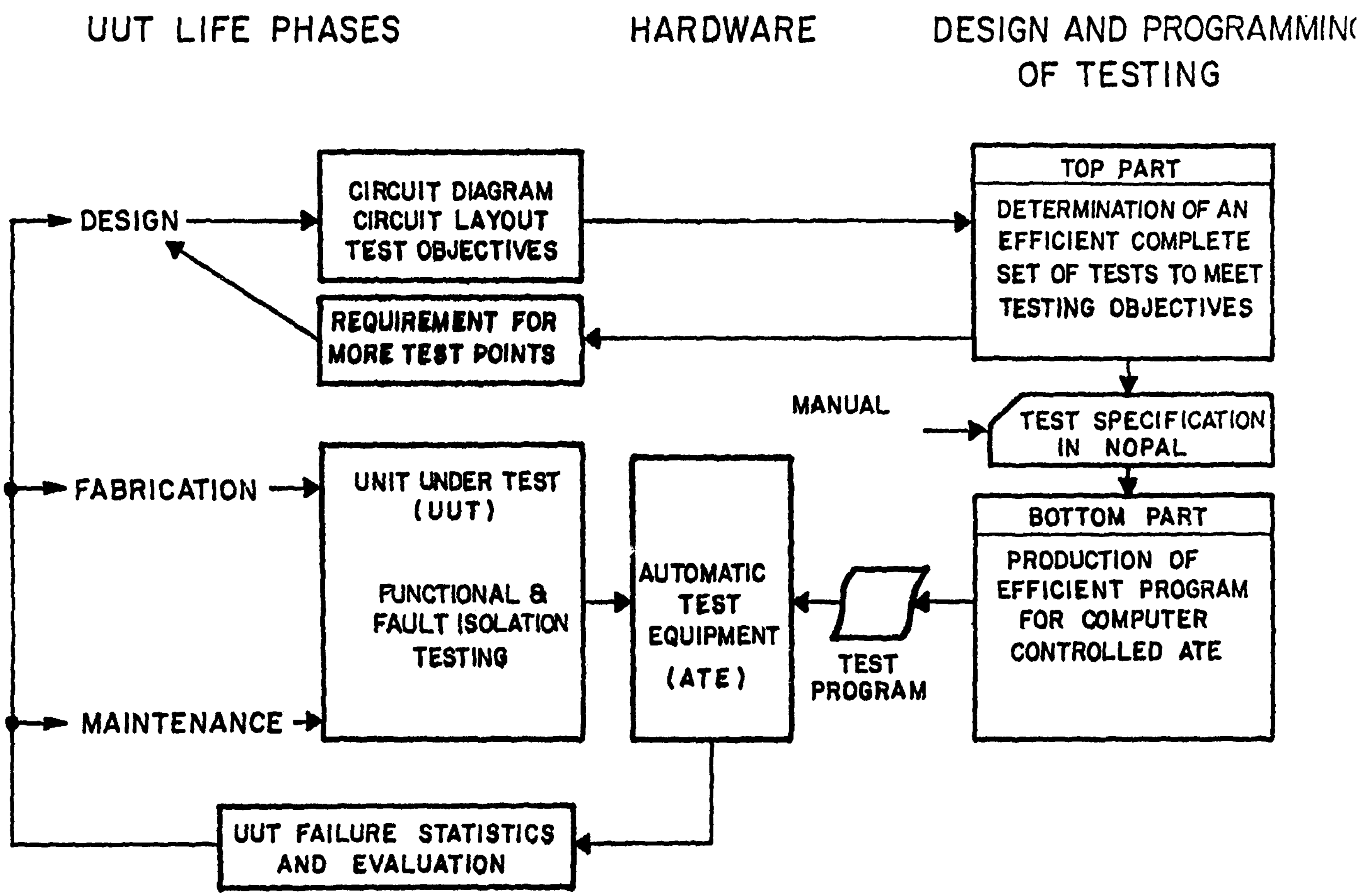

UUT LIFE PHASES
HARDWARE
DESIGN AND PROGRAMMIN(
OF TESTING
DESIGN
CIRCUIT DIAGRAM
CIRCUIT LAYOUT
TEST OBJECTIVES
TOP PART
DETERMINATION OF AN EFFICIENT COMPLETE SET OF TESTS TO MEET TESTING OBJECTIVES
REQUIREMENT FOR MORE TEST POINTS
MANUAL
TEST SPECIFICATION IN NOPAL
FABRICATION
MAINTENANCE
UNIT UNDER TEST (UUT)
FUNCTIONAL & FAULT ISOLATION TESTING
AUTOMATIC TEST EQUIPMENT (ATE)
TEST PROGRAM
BOTTOM PART
PRODUCTION OF EFFICIENT PROGRAM FOR COMPUTER CONTROLLED ATE
UUT FAILURE STATISTICS AND EVALUATION

FIGURE 1: DESIGN AND USE OF TESTING IN THE LIFE CYCLE OF A UNIT UNDER TEST (UUT)

The name NOPAL was selected for the system as it is the name of a cactaceous plant which illustrates in its growth of stems the incremental growth of tests during the life cycle of a UUT.

The plan for this article is to briefly describe the top and bottom parts in sections 2 and 3 respectively. The reader may refer to (Tinaztepe, 1977) for further description of the top-part, and to (Yung, 1977) for further description of the bottom-part.

The wide use of the NOPAL System is currently limited due to persistent software and hardware standardization problems which perplex the automatic testing field. Section 4 discusses approaches for resolving these problems.

2. Design of Testing: The Top-Part of NOPAL

The objective of this part of the system is to find a small and effective set of tests for a UUT, and express it in the NOPAL language. The process is illustrated in Fig. 2. The input required of the user shown on the left of Fig. 2 is described in Section 2.1. The methodology and process shown at the center of Fig. 2 are described in Section 2.2. The output reports, shown on the right of Fig. 2 are described in Section 2.3. The NOPAL language, in which the tests are specified, is described in Section 2.4.

2.1 Input to the Top Part of NOPAL

There are five input sections: (1) circuit description, (2) accessible test terminals, (3) UUT failure definitions, (4) fault-isolation testing-objectives and (5) measurement accuracy and initial conditions. The first is mandatory and the remaining are optional. They are all important to understanding the test design process.

Circuit Description of UUT: The test design is based on modelling and simulation of the UUT under nominal and malfunctioning conditions, to determine the symptoms of component failures. The modelling is based on the equivalent circuit drawing of the UUT provided by the designer. The equivalent circuit may consist of resistors, capacitors, inductors, mutual-inductances, voltage and current sources, bipolar and FET devices. Accurate modelling of the last two components may be an involved task, however sufficient published data is available for popular types. Each circuit component is given a unique name, where first letter identifies the component type. The value of any element may be defined by a numerical constant, table, or mathematical expression. Component tolerances must be specified by stating the maximum percentage deviation from the nominal value. Each circuit node is given a name. Current flow direction and source polarities are also indicated. The status of mechanical switches or potentiometers are treated as different

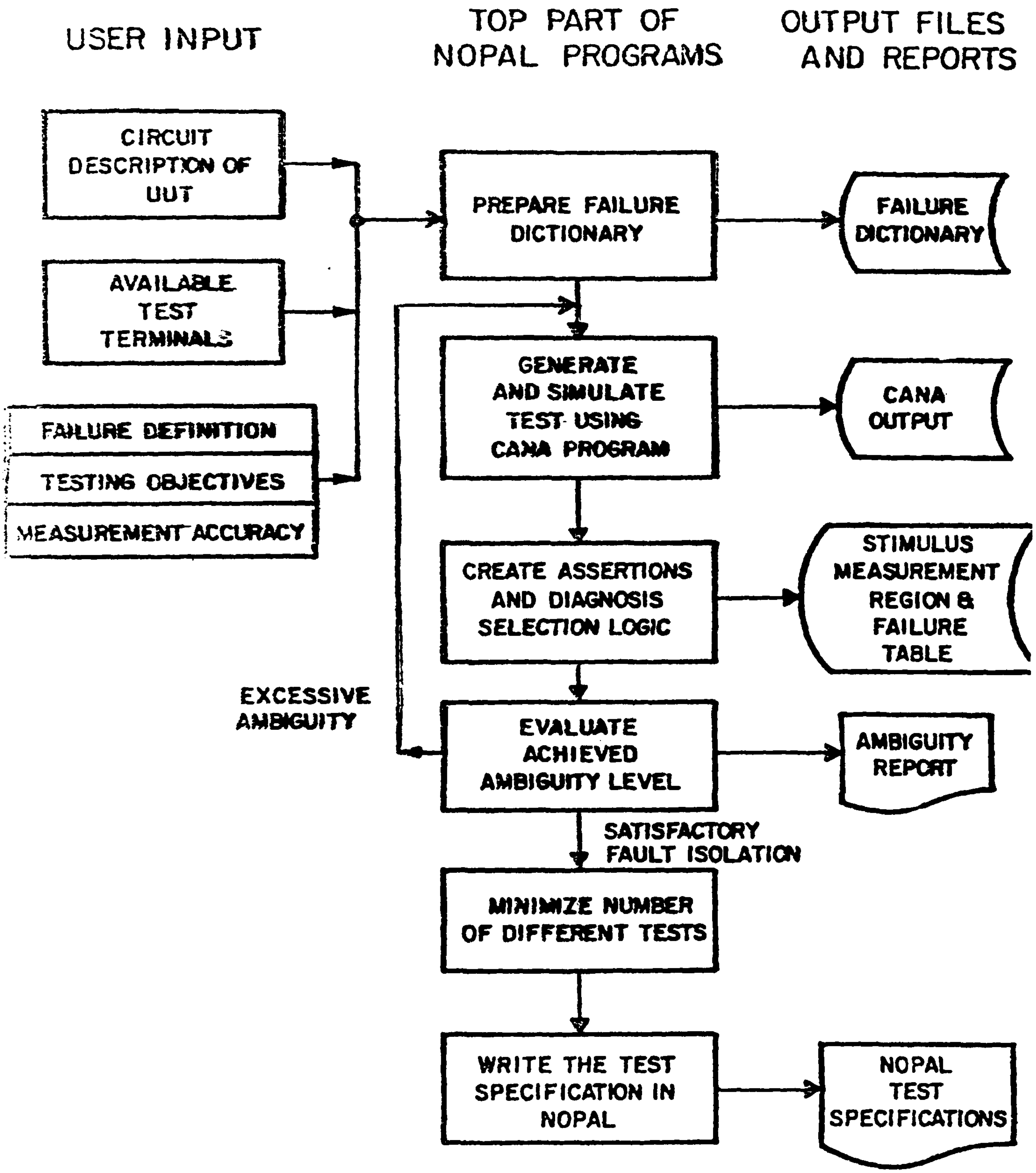

USER INPUT
TOP PART OF NOPAL PROGRAMS
OUTPUT FILES AND REPORTS
CIRCUIT DESCRIPTION OF UUT
AVAILABLE TEST TERMINALS
FAILURE DEFINITION
TESTING OBJECTIVES
MEASUREMENT ACCURACY
PREPARE FAILURE DICTIONARY
FAILURE DICTIONARY
GENERATE AND SIMULATE TEST USING CANA PROGRAM
CANA OUTPUT
CREATE ASSERTIONS AND DIAGNOSIS SELECTION LOGIC
STIMULUS MEASUREMENT REGION & FAILURE TABLE
EXCESSIVE AMBIGUITY
EVALUATE ACHIEVED AMBIGUITY LEVEL
AMBIGUITY REPORT
SATISFACTORY FAULT ISOLATION
MINIMIZE NUMBER OF DIFFERENT TESTS
WRITE THE TEST SPECIFICATION IN NOPAL
NOPAL TEST SPECIFICATIONS

FIGURE 2 FLOWCHART OF THE TOP PART OF NOPAL

initial conditions of the system. The description of an equivalent circuit follows conventions used in Computer Aided Circuit Analysis (CANA) programs. We use the NAP 2 CANA programs and follow its conventions. (Rubner-Peterson 1973).

Availability of Test Terminals: Any of the circuit nodes may be used for attaching test devices. However, the user may restrict the class of terminals available for testing.

Failure Definition: The objective of testing is to discover components of the UUT that have failed in a manner defined by the user. Since failure is a relative concept, any deviation from a component's nominal value may be declared by the user to be a failure. Failure definitions include topological changes, such as the removal or addition of a component. Typically catastrophic failures (open and short circuit) and out of tolerance are most common. To ease the tasks of input preparation, catastrophic failure modes are included automatically and the user has to declare only additional failure definitions. The number of tests that are required is related to the number of failures and testing may be extremely time consuming and expensive. The user can compromise between cost and quality by restricting the test objectives to discover only the more likely or most harmful failures.

Fault Isolation Test Objectives: To reduce the number of tests and lower testing costs the user may wish to accept tests which will sometimes not locate a failure in a specific component, butin one of the components belonging to a small group. The failure isolation requirement is expressed in statements denoting that P_k% of the total number of possible failures may be located in ambiguous classes consisting of k or less components. P's are cumulative percentages therefore $P_{k_1} < P_{k_2}$ ---- $k_1 < k_2$--k_n. For example, the user may require unique fault isolation (k_1=1) of no less than 50% (P_1=50) of the failures, and fault isolation to groups of no more than 4 components (k_2=4) in no less than 80% (P_2=80) of the failures.

Measurement Accuracy and Initial Conditions: Three types of accuracy may be specified: (1) minimum measurment threshold, (2) percentage accuracy of the measurement, (3) number of significant digits of the measured value. The default values are 0.1% measurement scale accuracy and four significant digits in the meter readings. All units are in MKS. The final optional input section specifies also the initial conditions of the UUT to speed the computer solution.

2.2 Methodology and Process of Test Design

As shown in Fig. 2, the first component of the process creates a failure dictionary data base for the UUT, based on the UUT circuit and failure descriptions supplied in the input.

Next, stimuli and measurements for tests are selected using the three strategies described bdlow, one at a time. First, small voltage stimuli are simulated as connected to connecting - points of the UUT, with the objective of measuring impedances at the connection points. This is referred to as the cold-circuit strategy. Next, the UUT is simulated as powered with the nominally specified d.c. power sources, and voltage and current measurements are conducted at nodes. This strategy is referred to as a d.c. - nominal. Finally, the circuit is simulated with a.c. signal at the input connecting-points and a.c. measurements at the outputs of the circuit. This strategy is referred to as a.c. - signal. The system design process provides for addition of more test strategies in the future. These strategies are all employed one at a time in the above order.

The circuit is simulated with the above stimuli, with the components having nominal values, and with the components having failure conditions enumerated in the failure dictionary, one at a time. The sensitivity of the circuit response due to tolerances is, also determined for each case. A CANA program, NAP 2 is used to perform the simulation. It has been selected based on considerations of economy of computer usage costs. The simulation produces for each test regions of values of physical entities, at each connecting point, corresponding to the variation in components within the allowed tolerance limits.

Based on this information, it is possible in the 4th component of Fig. 2 to verify if the tests formulated so far meet the test objectives set forth in the input. If testing objectives are still not met, the next strategy is employed, new tests are examined and the circuit is simulated until the above criteria is met, or until all three test strategies have been exhuasted.

When it is determined that the fault isolation test objectives are satisfied, the 5th component (see Fig. 2) is initiated. Its objective is to reduce the number of tests. It is necessary then to generate respective diagnoses. Next the effect of conjunctions and disjunctions of passing and failing tests is analyzed to improve fault isolation and reduce the number of tests. Redundant tests are then deleted.

The last component in Fig. 2 (6th component) produces the test specification in the NOPAL form, which is acceptable to the bottom-part of NOPAL.

2.3 Outputs From Test Design Process

There are three outputs produced by the top-part of NOPAL. The first output is an ambiguity report. The second is the test and diagnosis logic in tabular form. Finally, a test specification in NOPAL is produced.

Ambiguity Report: The ambiguity report consists of percentages P_k% and number of members - k for each class (see Test Objectives in Section 2.1). It also identifies the actual failure members in each class.

Test and Diagnoses: This report aids human review of the tests and the selected diagnoses.

NOPAL Test Specification Report: This report consists of a listing of the test moduler, which is one of the components of the specifications. The other components, the ATE and UUT specifications, must be completed manually and used as input to the bottom part of the NOPAL system. (See discussion below of the NOPAL language).

2.4 The NOPAL Language

NOPAL statements can appear in any order due to the nature of non-proceduralness. Yet for organization purposes, each test specification is divided into the UUT, ATE and Test-Module Sections.

UUT Specification: The UUT-oriented information is grouped into two sections: (1) UUT connecting points which identify the connecting pins for the testing and (2) component failures which optionally specify potential faulty components with failure modes (i.e., types of failures). Protective limits can be specified to protect the connecting point from inadvertent damage caused by excessive stimuli power. A failure index is used to grade the components by their likelihood of failure. This information is used to increase the execution efficiency by first testing the components which are more likely to fail.

ATE Specification: ATE related information which is needed to verify the test modules and UUT specifications is organized in two sections: (1) ATE connecting points, which are connected to the matching UUT connectors, and (2) ATE functions, which specify the stimuli devices, and the types of component failures. Purely computational functions may also be used and listed here.

Test Module Specification: This section includes a collection of test modules and the definitions of the diagnoses and messages which are referenced in any test module. The test modules specification is the core of the NOPAL specification. Each test module is specified independently of the others. Thereby individual test modules can be modified, deleted, or added without affecting the rest of the test modules. The subparts of a test module (in addition to the test module label) are: (1) the stimuli that need to be applied to the UUT at test time, (2) the measurements that n-ed to be made with the comparisons that will determine the results, and (3) the logic that selects diagnoses based on the results.

3. Automatic Program Production: The Bottom Part of NOPAL

Fig. 3 illustrates the components of the automatic program production system. The inputs are test specifications written in NOPAL.

The first component in Fig. 3 performs syntax analysis of the test specification. Also, the test specification is encoded and stored in a simulated associative memory to facilitate later processing. Syntactical errors and documentation consisting of a specification listing and several cross reference reports, formatted for easy readability, are produced.

The second component incorporates an engineering knowledge-base needed to determine and optimize the sequence of execution. In the course of analysis, the system produces various additional reports including error/warnings of detected inconsistencies, fault locating summary, and a flowchart showing the test execution sequence.

The third component generates a test program in EQUATE ATLAS acceptable to an RCA AN/USM-410 series ATE. The object program will be compiled by the EQUATE ATLAS compiler, and then be ready to test the given class of UUT's.

The automatic sequencing and optimization process is further discussed below because of its importance and novelty. The NOPAL system automatically optimizes intra-test and inter-test execution sequences and generates control logic for dynamically evaluating the conditions that determine the progress of the testing and selecting the next test during execution. In the process of sequencing, the test modules and their subcomponents are considered each as an integral unit that may be represented by a node in a directed graph. The specification is analyzed to determine precedence relationships between test modules or their subcomponents. These precedence relationships can be represented by directed edges in the directed graph. In all these precedence relationships, the former node must precede the latter at execution time of the object test program and is said to be a predecessor of the latter, while the latter is said to be successor of the former. The six major relationships are briefly explained in the following.

(1) Data determinacy incorporates the principle that data must be generated before it can be used. The generation of data by a predecessor test module is recognized by the declaration of a TARGET variable. A successor test module references the same variable, declared as SOURCE.

(2) Interactiveness relationships are dictated by the need to exchange messages interactively with the ATE operator. Its predecessor is a diagnosis, and successor a test module.

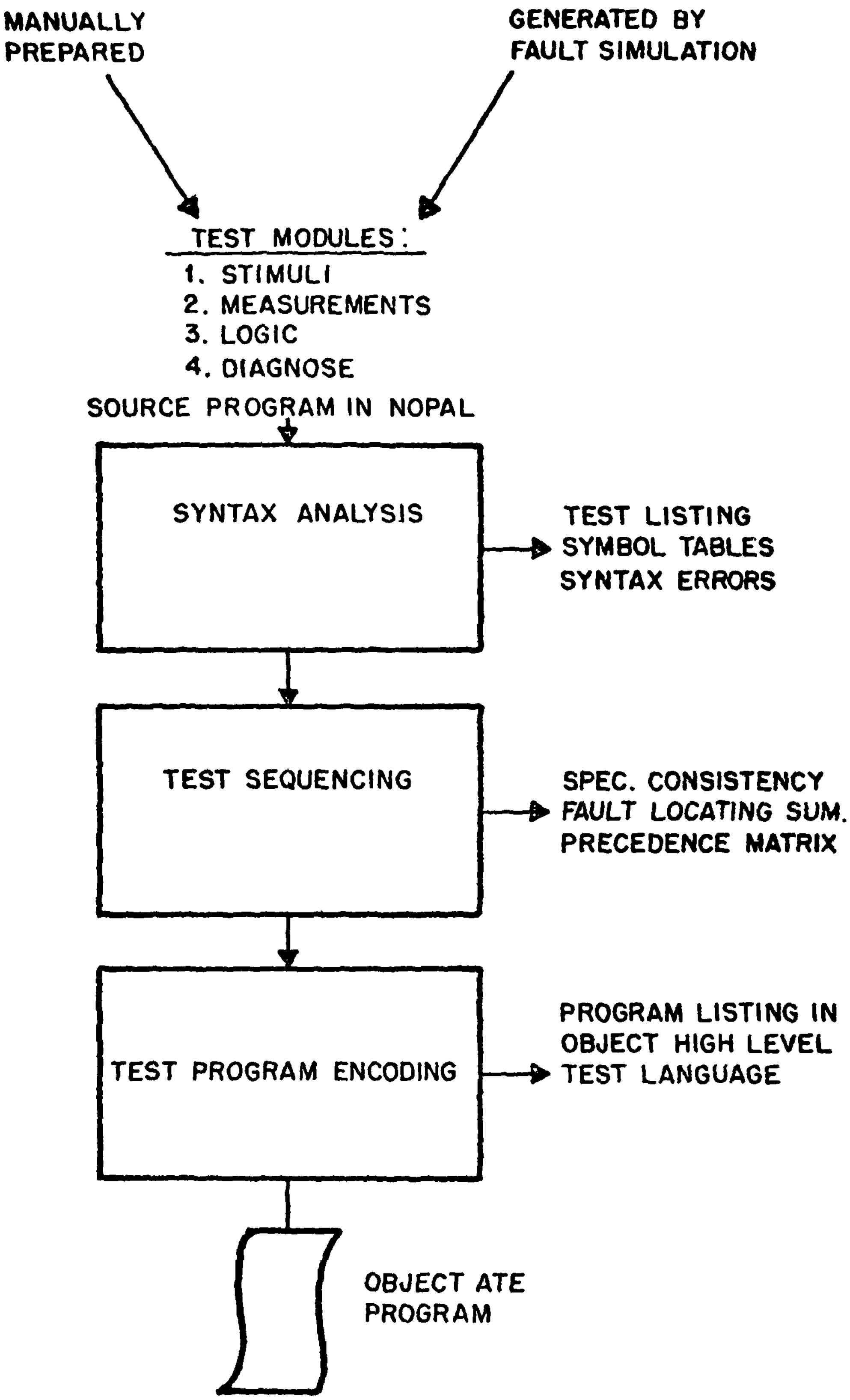


FIGURE 3 COMPONENT OF AUTOMATIC PROGRAM GENERATING SYSTEM

(3) Component protection is based on the concept that non-destructive testing can be achieved if a critical component is tested before other components which depend on it for their normal operation. Hence, the failure of such a critical component will prohibit further testing for those dependent components.

(4) Fault isolation strategy schedules tests in a top-down fashion using component subset relationships. The more generic fault isolation tests are performed first. The lower level, more specific tests are then executed or skipped, depending upon whether the failure is detected at the top level.

(5) Stimuli application is concerned with efficient application of waveform stimuli. It is based on the assumption that application of stimuli is most time consuming, hence it is advisable to conduct all the possible tests once a stimulus is applied.

(6) Failure likelihood uses the idea that efficiency is obtained by first testing those components which are more likely to fail. Information is extracted from the failure index field in the UUT Component Failures specification.

Based on the graph, the consistency, completeness, ambiguity, and feasibility of the test specifications may be checked. Possible cycles in the directed graph imply errors. They are detected and reported to the user. Finally, all nodes are ordered in proper execution sequence making up a flowchart of the program.

4. Impact of Lack of Test Software and Hardware Standards on the NOPAL System

As noted previously, the present use of the NOPAL System is limited to the RCA EQUATE AN/USM-410 automatic test equipment and the EQUATE ATLAS compiler that is associated with the equipment. This limitation is due to lack of hardware and software standardization in the field of automatic testing. There are two main aspects to this problem.

First, there is lack of a common high level test programming language that can be used with different types and models of automatic test equipment. Although the names of the languages used, such as BASIC, FORTRAN, and ATLAS, are common, in fact the compilers produced by various manufacturers of automatic test equipment vary greatly, even for the same named languages. For instance, the RCA EQUATE ATLAS language, in which the NOPAL system produces programs, differs greatly from the Hewlitt Packard ATLAS language. The great efforts that have been made to develop software standardization in the field of automatic testing are partly responsible for the chaos in this area. The ATLAS and the IEEE standards committees published an ATLAS standard in 1977. But the newly proclaimed standard is admittedly not a programming language for which a compiler may be constructed. More recently a subcommittee of the ATLAS committee has been considering issuing a new standard, named ATLAS-90, for which it would be feasible to construct a compiler. However, the prospect of a usable and proven compiler for a standard test programming language appears still far off. The US Army, cognizant of this problem, has developed a proposed standard test programming language named OPAL. Although OPAL has a potential for greatly improved effectiveness for test programming, and a compiler has not yet been constructed for it.

The second aspect of the problem is the lack of standardization in the stimuli and measuring devices utilized in the various automatic test equipment. This means that a programmer must know intimately the test devices and a test program composed in a high level language must be oriented to a specific set of stimuli and measuring devices. Namely, even if there was a common high level test language, the programs would still not be portable from one type or model of automatic test equipment to another. This restriction would hold even where another model of automatic test equipment contains similar or equivalent stimuli and measuring devices. The OPAL language has facilities to specify the parameters of stimuli and measuring devices that are utilized in a program. It is envisaged that OPAL compilers for respective models of automatic test equipment would be able to assign the available devices that correspond to those specified in the program.

Because of this situation, it was necessary to incorporate in NOPAL various facilities that would facilitate its wider use. There is a need for two types of capabilities; first, to produce test programs in a variety of high level test languages, and second, to incorporate in the produced programs use of stimuli and measuring devices that are available in the automatic test equipment to which the produced programs are oriented. The facilities to attain these two capabilities in NOPAL are as follows:

(1) All except one of the NOPAL system components are independent of the object high level test language in which the program are to be produced. The only component of the system that is dependent on the test programming language is the Code Generation component shown at the bottom of Fig. 3. Furthermore, this component incorporates tables which translate the entries in the flowchart (produced by the Test Sequencing component shown in Fig. 3) into respective test programming language statements. To produce programs in a language different than the EQUATE ATLAS, it would be necessary only to modify the code generation component of the system. It is hoped that the modifications would not be difficult due to the tabular structure of this component.

(2) As already noted in the discussion of the NOPAL language, the NOPAL specification has an ATE section where stimuli and measuring devices that are to be utilized may be described. The NOPAL system

includes a library of routines, in the object test programming language, which correspond to the devices specified in the ATE section of the NOPAL specification. Thus, use of additional or different devices may be incorporated in the program by entering in the library of the NOPAL system routines that employ these devices. This feature also allows the use in the NOPAL specification of very high level and complex devices, which in fact require a number of lower level real devices to perform the equivalent function. This capability allows the use of higher level statements, thereby saving much labor by the user.

References

T. Rubner-Pieterson, "Network Analysis Program NAP 2", Technical University of Denmark, Lyngby, Denmark, March 1973.

C. Tinaztepe, "Automatic Test Design", Ph.D. Dissertation in Computer and Information Science, University of Pennsylvania, 1977. Also, Research and Development Report ECOM-75-0650-F2, US Army Electronics Command, June 1978.

Y. Chang, "Automatic Test Program Generation", Ph.D. Dissertation in Computer and Information Science, University of Pennsylvania, 1977. Also, Research and Development Report ECOM-75-0650-F-1, US Army Electronics Command, March 1978.

"IEEE/ARINC Standard ATLAS Test Language 416-1976", IEEE Inc., 1972.

"Military Standard, Operational Performance Analysis Language (OPAL)", MIL-STD-1462A, US Army Research and Development Command, March 1978.

Some Issues Related to Computer System Built-in Test

J. B. Clary
A. R. Jai

Research Triangle Institute

As system complexity has increased, attention has focused on the need for military electronic systems which are easier to maintain. Modular systems which have an integral fault detection and isolation capability, referred to as built-in test (BIT), have been explored in the past as a means of reducing repair costs while providing continuous system performance monitoring. However, as circuits have become more highly integrated and system complexity increased, the ability of system users to verify performance and to rapidly locate faulty modules has been severely affected. This has been particularly apparent in programmable digital systems.

This paper addressed some of the important issues related to the design of built-in tests in programmable modular digital computers such as the Military Computer Family. These issues included: (1) identification of system on-line performance monitoring requirements, (2) identification of relevant fault communication to the appropriate level within the system hierarchy, and (3), identification and application of BIT effectiveness measures to quantiatively assess the candidate test strategies.

SOME ISSUES RELATED TO COMPUTER SYSTEM BUILT-IN-TEST

J. B. Clary
A. R. Jai

Research Triangle Institute
Research Triangle Park, N. C. 27703

I. BACKGROUND

As system complexity has increased, attention has focused on the need for military electronic systems which are easier to maintain. Modular systems have evolved as a way of meeting this need by offering system users a means by which complex electronic systems may be rapidly repaired if the faulty modules can be identified [1], [2]. However, as circuits have become more highly integrated and system complexities have increased, the ability of system users to verify system performance and to rapidly locate faulty modules has been severely affected [3], [4], [5]. The need, therefore, presently exists for improved approaches to modular system fault determination and resolution.

One approach which has received some attention in recent years is the use of built-in-test (BIT) for on-line performance monitoring [6]. In the case of modular system structures, this concept requires the incorporation of fault monitoring as an integral part of basic module designs. Implicit in the fundamental idea of BIT is the need for designers to take some action to ensure that new systems have essential fault detection capabilities. This need, in turn, requires that designers understand technically sound BIT techniques as well as reliable analytical methods which may be used to assess alternative BIT approaches.

Unfortunately, there are few places in the realm of modern system design and development where ad hoc approaches are more apparent than in the present day fault detection, isolation and repair area. More often than not, decisions concerning the kinds of fault monitoring which should be implemented are left entirely up to individual circuit and software designers. These designers frequently lack the detailed technical knowledge as well as the motivation for doing a thorough job of incorporating adequate performance monitoring at all system levels. Obviously, ad hoc approaches to built-in-test are counter to the nature of otherwise orderly modern system design processes.

This work was supported in part by Contract N00039-77-C-0363 from the Naval Electronics System Command and Contract DAAG29-76-D-0199, D. O. No. 0781 from the Army Communications Research and Development Command.

The need for an orderly approach to integral fault detection and reporting is even more apparent in software programmable digital systems. There are a number of reasons why this is true, the most apparent of which are:

1. The additional flexibility made possible by machine programmability requires more versatile fault detection approaches,
2. The same programmable structure may be used in real-time as well as non-real-time operational environments which may preclude off-line software-only test schemes,
3. The presence of an internal machine hardware/software interface which does not exist in non-programmable hardware,
4. The opportunities which exist in many applications of programmable digital systems for sophisticated error recovery schemes.

The existence of the need for more orderly approaches to integral fault detection, reporting and error handling reflects the way modern computers have evolved. Programmable digital machines have evolved in such a way that areas of technical specialization have tended to center about either digital hardware or software. As a consequence, digital systems specialists often view problems from either a hardware standpoint or a software standpoint. Communication between the two schools, in many instances, has been minimal.

Another important consideration is that while commercial minicomputer functional capabilities often may be adequate to meet the needs of the military, there can be differences in on-line performance monitoring and fault localization philosophies. Special attention must be given to the military emphasis on minimizing total system life cycle cost (LCC) as opposed to emphasizing the minimization of purchase price.

The Army, through the Center for Tactical Computer Systems (CENTACS), Office of the Communications Research and Development Command (CORADCOM), is working cooperatively with the Navy and the Air Force on a new approach to developing and acquiring computers for the military [7 - 10]. The effort is known as the Military Computer Family (MCF) Program. The Military Computer Family Program goal is to provide defense system developers with a software-compatible family of standard, modular computers. The MCF program stresses software compatibility between prior generation computers at the level a programmer needs to know to write time-independent machine language programs. At the same time, the MCF concept calls for hardware module compatibility through standardized Form, Fit and Function (F^3) specifications.

A consequence of being able to use existing software and state-of-the-art modular hardware is the potential reduction in total computer system life cycle cost. An important aspect of the proposed MCF procurement procedure with potential LCC savings is the concept of hardware vendor warranties. The MCF hardware warranty concept is viewed as a means of reducing logistic support costs through improved reliability. Implicit in this approach is the necessity of 1) knowing with a high degree of confidence that a module is not performing properly, 2) identifying which module is faulty, and 3) effecting repair through module replacement. To meet this need, an effective and efficient means of detecting and locating hardware faults is necessary.

The following discussion considers some important issues relevant to detecting and reporting faults in programmable modular digital computers such as the Military Computer Family.

II. RELEVANT BUILT-IN-TEST ISSUES

The on-line performance monitoring problem is characterized by the need to achieve effective observability without interfering with the functional processes under observation. Implicit in this requirement is the need to add redundancy in some form to the system. This redundancy may take the form of additional hardware, firmware, software or combinations of all of these. It is essential to be able to observe appropriate parameters, make decisions concerning these observations and report the findings to an appropriate level within the system hierarchy. To do so effectively implies the introduction of test performance and cost metrics based upon system objectives and knowledge of the system fault population. Based upon an understanding of these requirements, intelligent decisions can be made concerning the allocation of built-in-test resources within the functional hardware and software structure.

To summarize, the incorporation of built-in-test involves the following issues.

1. Identification of system on-line performance monitoring requirements,
2. Identification of the fault population,
3. The determination of a strategy for allocating the built-in-test resources including fault communication to the appropriate level within the system hierarchy,
4. The identification and application of built-in-test effectiveness measures to quantitatively assess candidate test strategies.

The following discussion considers each of these issues in further detail. In order to be specific, the Military Computer Family is used as an example of the incorporation of built-in-test in programmable computers.

On-Line Performance Monitoring Requirements

The Military Computer Family program addresses the need within DOD for new generation digital hardware while maintaining software compatibility with prior generation machines. In addition, the MCF concept goes beyond the transportability of software from old machines to new.

Under the MCF program, a whole new procurement process is made possible through modular hardware computer structures. In particular, the modular hardware structure of MCF allows components to be procured simultaneously from multiple sources.

In order to insure the success of this competitive procurement process and to motivate vendors to produce reliable form, fit and function compatible designs, MCF components will be procured with vendor-backed warranties. The MCF warranty concept provides incentives for vendors to produce reliable components. At the same time, however, it places the burden of identifying faulty modules with a very high degree of confidence on the government in order for warranties to be exercised. It is, therefore, essential that available means be provided for identifying and locating faulty MCF components. An additional aspect of the MCF warranty concept is the necessity of measuring component on-time if the components are to be warrantied for a pre-determined number of hours. In addition to being important for warranty validation, the elapsed time data may be useful in fault prediction and localization.

Finally, it is vital for MCF users to know system operating capabilities and limitations at all times. On-line performance monitoring is of utmost importance to field commanders and others who rely on computer systems to provide them with combat-critical information. It is, therefore, a primary objective of BIT for MCF to meet this requirement through timely fault detection and alerting.

In summary, the objectives of BIT for MCF are:

1. To provide continuous system monitoring and indication of system malfunction,
2. To diagnose the cause of system malfunction to a module level with a low probability of a false module pull, and
3. To measure and record module elapsed power-on time.

Fault Population

Hardware faults in digital computer systems can be classified in two basic categories: the stuck-at (solid or permanent) faults and the intermittent (transient) faults. The stuck-at faults occur when a logic signal remains permanently in either a one or a zero state. Such failures are consistent and the failure symptoms are reproducible. This facilitates the isolation of stuck-at faults through well defined diagnostic test procedures.

The intermittent faults on the other hand are defined as random failures that prevent the proper operation of a unit for a short period implying that the duration of the failures is not long enough for the application of a test procedure designed for permanent faults [11]. The intermittent faults occur due to environmental as well as non-environmental reasons. Environmental conditions such as temperature, humidity, vibration, electrical and electromagnetic interferences, etc., induce intermittent faults. More important, however, are the non-environmental intermittent faults which are caused by loose connections, resistance variations, deteriorating or aging components, etc.

Recent studies in fault diagnosis [11], [12], [13] indicate that a major portion of digital system malfunctions are caused by intermittent faults. In some systems, 80 to 90 percent of the faults are estimated to be intermittent [11]. Furthermore, these faults have been found to account for more than 90 percent of the total maintenance expense because they are difficult to detect and isolate.

Built-In-Test Resource Allocation

In view of the built-in-test objectives and the Form. Fit, and Function (F^3) specifications for the MCF computers, a top-down approach to the allocation of BIT resources has been recommended. In the MCF computers, the following three hierarchical levels are easily identifiable.

1. MCF Member Level (System Level)
2. Chassis Level
3. Module Level

Built-In-Test techniques may be incorporated at any one or combinations of the above mentioned hierarchical levels. The basic approach used in this study is to identify a set of BIT techniques at each level and then select candidate BIT techniques based on certain performance versus cost criteria. The BIT effectiveness criteria for performance and cost are discussed later.

Each hierarchical level affords a certain level of fault detection and a degree of fault isolation capability because of the observability and controllability problems. In order to enhance the performance/cost figure of the candidate BIT techniques, it is necessary to study the fault detection requirements and the BIT resources available at each hierarchical level. Furthermore, the fault communication and hardware/software interfaces between the various constituent BIT elements at each hierarchical level need to be investigated.

In summary, fault detection and identification at the various levels may be performed using continuous monitoring, sampled monitoring, idle time monitoring or other off-line techniques. Approaches should be emphasized which provide continuous monitoring with minimum impact on system performance.

Built-In-Test Fault Communication

An important consideration in designing programmable computers with built-in-test is the communication of the fault information within the system. The fault information generated by the on-line monitoring hardware passes through several levels of hardware and software ultimately to the user. A systematic means of communicating the fault information is essential to facilitate the handling of errors in an orderly fashion.

In modular computers, such as the MCF computers, the hardware is functionally partitioned into modules which communicate with each other via common bus structures. A collection of modules which complement each other functionally form a hardware subsystem. The complete system consists of a number of subsystems. This type of partitioned hardware structure lends itself to a similar partitioning in the implementation of the built-in-tests for on-line fault monitoring. In this case, some fault detection circuitry would reside on each module and monitor the operation of that module.

The faults when detected must be reported from the module level to the subsystem level and ultimately to the system level. The fault communication may occur over existing system bus or may be done via separate fault monitoring bus. In either case, it is necessary to preserve the identify of the fault source, the type of error, and the state of the machine at the time of the occurrence of the faults. This may be done in hardware by using status bits or register(s) associated with every module or subsystem.

In addition, the occurrence of the fault condition must be reported to the next higher level at the earliest moment so that quick response for error recovery is possible and the effect of fault on the rest of the system can be restricted. Error recovery may be attempted in the hardware at the level at which the fault is detected or at succeedingly higher levels by retrying partially or completely the instruction cycle in which the fault was detected. Instruction retry presumes that the state of the hardware at the beginning of the cycle is preserved.

If the hardware recovery is either not possible or the attempts at recovery fail due to the existence of a permanent fault at any hardware levels, the fault information should be communicated to user via the software.

The hardware/software interaction in the above context of fault communication and error handling is important because it addresses the fundamental issue of fault observability in programmable computers from the user standpoint. Before discussing the BIT hardware/software interface, it is important to understand the process of fault reporting in software.

The functional software controls the functional hardware by a sequence of instructions to the hardware. In most hierarchically structured software systems there are several levels of software. The highest level is the application software which is closest to the user. The lower levels consist of specific task oriented routines which are called from the application software. There are two methods available for reporting faults to higher levels. These are discussed below.

The first method delegates to each level the responsibility of acquiring fault information and reporting it to the next higher level. This involves the use of error termination code at all levels. Each called program has at least one returned error parameter. This indicates, after the termination of the program whether an error has occurred during the execution of the program. This may be accomplished by checking the appropriate hardware error status registers during the execution of the program.

This approach is generally used for reporting errors which are non-critical to the operation of the system, such as I/O errors. An advantage here is that it permits the handling of errors at the proper level which has enough information to take appropriate action. There are several drawbacks in this approach. Firstly, it makes the programs more complex; secondly, it makes the separation of functional program and error handling routines more difficult. Furthermore, there is a delay of at least several instruction cycles between the time fault is detected and the time it is reported to a level where corrective action can be taken.

An alternative approach is to use interrupt or trap mechanism to report faults. Here the normal execution of the current instruction sequence is terminated and control is transferred to a predetermined location at the occurrence of the fault. Flexibility may be provided by making the trap vectors user specifiable. Also, a specific trap location may be associated with a fault or a group of related faults which preserves their identity in the reporting mechanism. Thus, separate trap handling routines may be provided at each trap location to specifically handle that type or class of faults. Trap handling routines can, if possible, begin immediate error recovery otherwise notify the user directly of the error condition. Trap mechanism is generally used for reporting critical faults such as those in the memory or CPU which demand immediate attention.

The above discussion leads to a possible conceptual BIT hardware/ software interface. The BIT hardware/software interface consists of a hardware module, the BIT-status hardware module (BHM) and a software (or firmware) module, the BIT-status software module (BSM). These two modules, acting in concert, maintain the fault status of the computer hardware and provide the ability to invoke appropriate handlers for hardware faults as they occur. Figure 1 depicts the overall framework in which these modules reside [14].

The function of the BHM is to receive the Pass/Fail (P/F) indicators generated by the BIT circuitry associated with the functional hardware modules. Should a fail condition be indicated, the BHM module will register this failure according to the module indicating the fault and will, at the same time, generate an (high-priority) interrupt to the functional hardware. Recognition of the interrupt by the functional hardware will cause control to be transferred to the BSM, the first in a chain of software routines which will handle the fault.

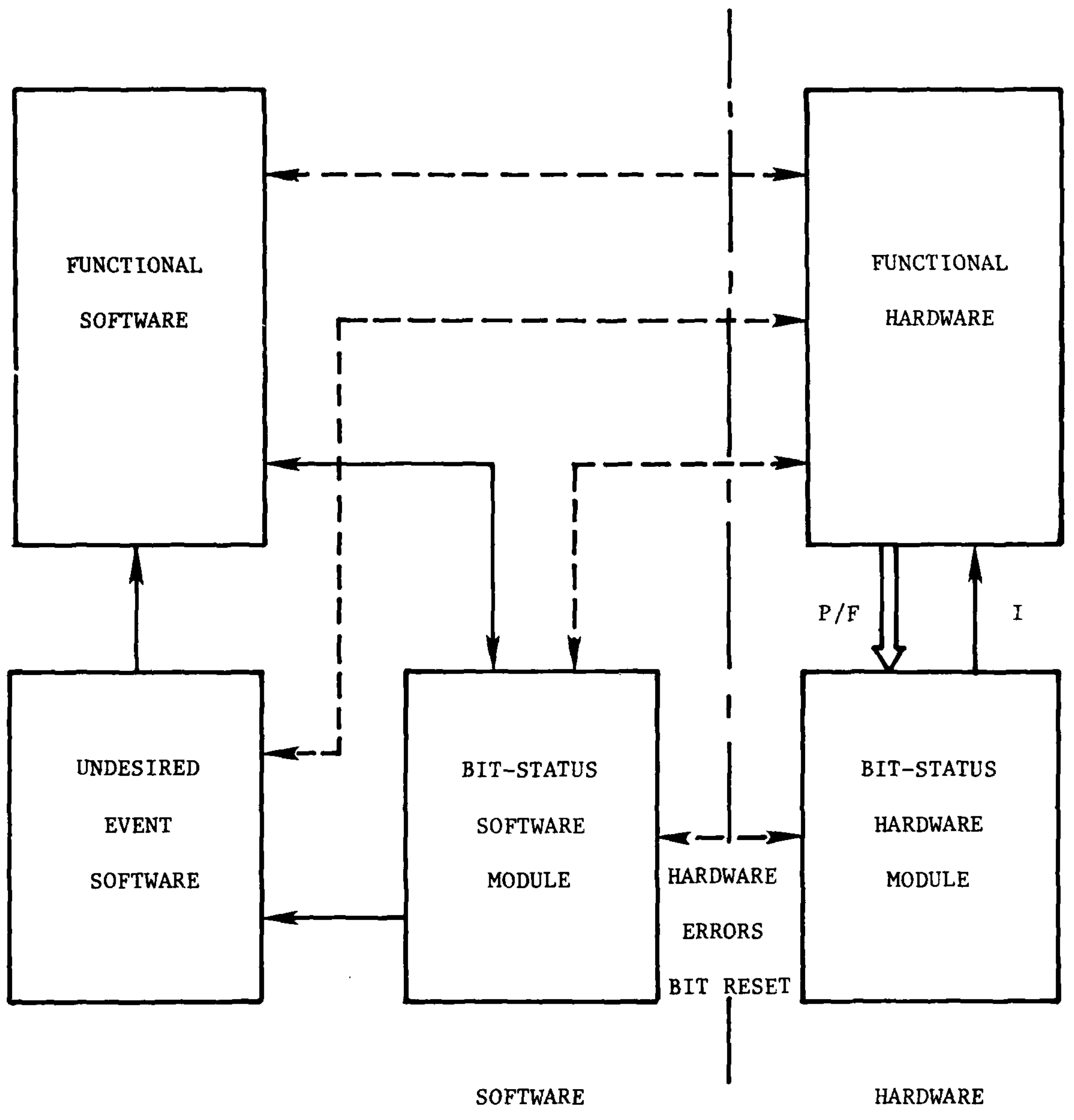


Figure 1 Programmable Computer With Hardware Built-In-Test [14]

It is envisioned that the BSM will be a general-purpose system routine, adapted to the computer and its general application, and not to specific users of the computer. The BSM, therefore, will not (in general) be part of the user-supplied application software and may appear in the form of firmware.

Built-In-Test Effectiveness Criteria for Performance and Cost

The effectiveness of any built-in-test approach may be measured in terms of the ratio of its performance to the cost of implementing it. In quantifying the performance/cost ratio there are a significant number of parameters or sets of parameters which can be considered.

In view of the broad objectives of the built-in-tests for the MCF computer systems, a set of general parameters has been chosen. Since the main objective of the BIT for the MCF is to detect and isolate faults with a low probability of false module pull, the performance parameters should be able to measure the probabilities of detecting and localizing faults as well as the probabilities of false alarms. Furthermore, since the mean time to repair is also an essential consideration in the maintenance of MCF computer systems, the performance parameters should include the time required to detect and isolate faults. This forms a set of five performance measurement parameters which are defined below.

P_{SFD} - Probability of System Fault Detection
P_{LFE} - Probability of Localizing to Faulty Element
P_{FA} - Probability that suspected Faulty Element is not Faulty. (False Alarm)
T_{SFD} - Time to System Fault Detection
T_{LFE} - Time to Localize to Faulty Element

The cost of implementing a BIT approach can be broadly categorized into hardware and software costs. The hardware costs mainly involve space, power, and failure rate. The hardware cost can be measured in terms of the percent increase in the space, power, and failure rate due to the additional BIT circuitry.

The software costs on the other hand are more difficult to assess. The software is impacted at three levels: 1) operating system software, 2) applications software, and 3) diagnostic software. Additional BIT functions typically increases the operating system responsibilities because it must provide for the user/BIT interface and may have to perform error handling tasks. The BIT functions are generally transparent to the user. However, the application software will be impacted if the user is to be provided with the option to control some of the BIT functions. The diagnostic software can generally be simplified by additional BIT hardware.

III. CONCLUSIONS

The preceding sections have briefly discussed several issues relevant to the incorporation of built-in-test resources in programmable digital computers. These issues included 1) identification of performance monitoring requirements, 2) identification of the relevant fault population, 3) the determination of a strategy for allocating built-in-test resources within the system hierarchy, 4) the identification and application of built-in-test effectiveness measures upon which to quantitatively assess candidate test strategies. One of the important aspects of any BIT approach in programmable digital systems is the communication of fault information between hardware and software. Primary elements of the interface between BIT hardware and error handling software have been identified and ways to do task allocation between hardware and software have been considered.

REFERENCES

1. "Military Standard Design Requirements for Standard Hardware Program Electronic Modules," MIL-STD-1389 (Navy), March 14, 1973.

2. "The Navy Standard Hardware Program," NAVELEX 0202-05Aa, November 1973.

3. "Electronics-X: A Study of Military Electronics With Particular Reference to Cost and Reliability." Volume 2. Final Report for Defense Advanced Research Projects Agency. Prepared by Institute for Defense Analysis, Arlington, Virginia, 1974.

4. "Report on Navy Issues Concerning Automatic Test, Monitoring and Diagnostics System and Equipment." Prepared for the Assistant Secretary of the Navy (R&D). 13 February 1976.

5. "Electronics Technology Exploratory Development (ED) Strategy," Submitted to Director of Navy Technology, 28 September 1976.

6. Clary, J. B., Gault, J. W., Weikel, S. J., Whisnant, R. A., Alberts, R. D., "A Study of a Standard BIT Circuit," Final Report, Contract N000163-76-C-0231, Naval Avionics Facility, Indianapolis (NAFI), Prepared by Research Triangle Institute, Research Triangle Park, N.C., February 1977.

7. Coleman, A. H. and W. R. Smith, "The Military Computer Family: A New Joint-Service Approach to Military Computer Acquisition," Computer, Vol. 10, No. 10, October, 1977.

8. Barbacci, M. R. and D. P. Siewiorek, "Evaluation of CFA Test Programs via Formal Computer Descriptions," Computer, Vol. 10, No. 10, October, 1977.

9. Burr, W. E. and R. Gordon, "Selecting a Military Computer Architecture," Computer, Vol. 10, No. 10, October, 1977.

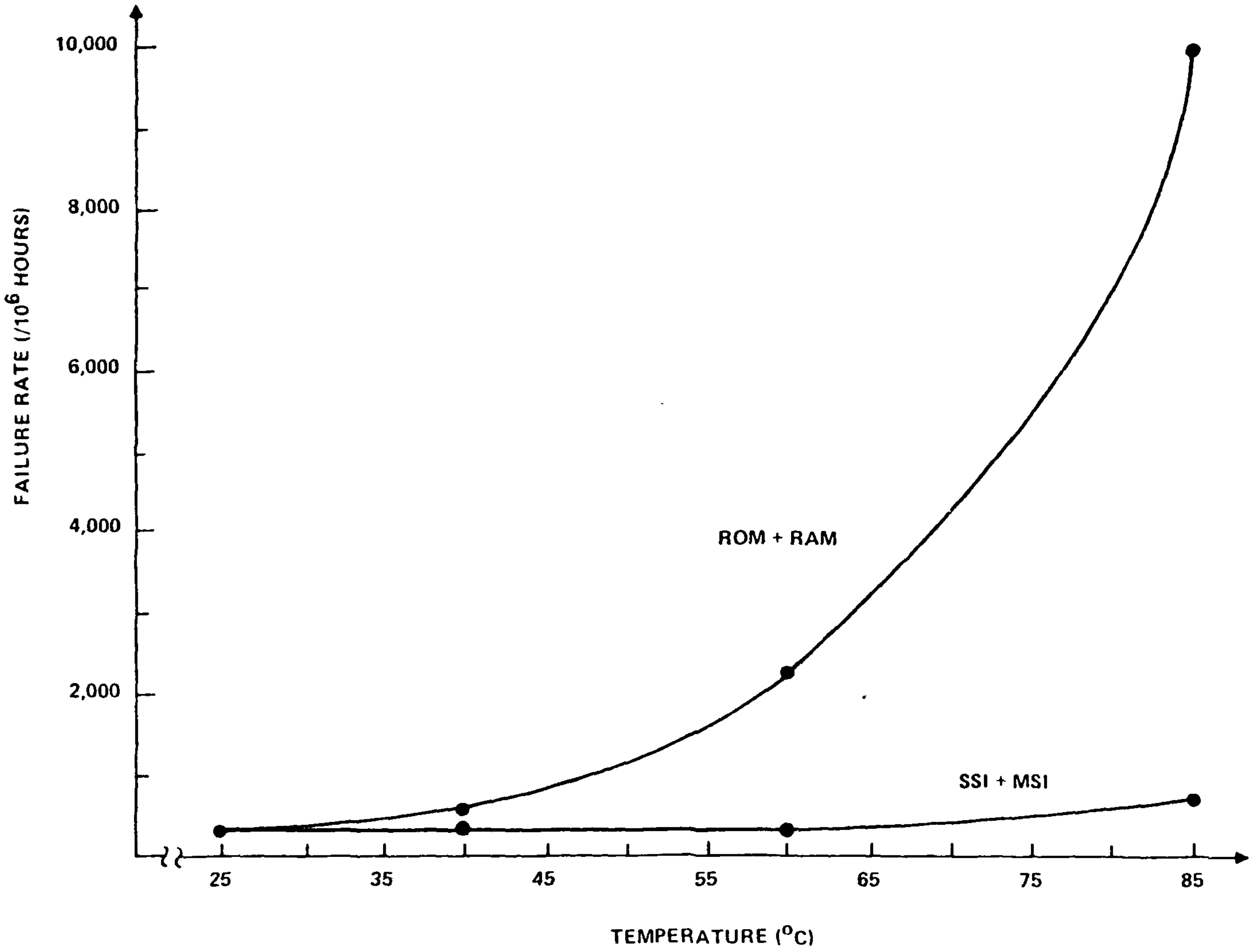
10. Fuller, S. M. and W. E. Burr, "Measurement and Evaluation of Alternative Architectures," Computer, Vol. 10, No. 10, October, 1977.

11. Tasar, O. and V. Tasar, "A Study of Intermittent Faults in Digital Computers," Proceeding of National Computer Conference, 1977.

12. Maestri, G. H., "The Retryable Processor," Proceedings of Fall Joint Computer Conference, 1972.

13. Ball, M. and F. Hardie, "Effect of Detection of Intermittent Failure in Digital Systems," Proceeding of Fall Joint Conference, 1969.

14. Clary, J. B., Haidt, J. G., Parnas, D. L., "Basic Research in Support of Concurrent Fault Monitoring in Modular Digital Systems," Section 3.1 of Interim Technical Report, Contract N00039-77-C-0363, Naval Electronics Command, Code 304, Washington, D.C., Prepared by Research Triangle Institute, Research Triangle Park, N.C., January, 1978.



FAILURE RATE AS A FUNCTION OF TEMPERATURE FOR THE PDP 11/70

FAILURE RATE OF PDP-11/70 at 25°C

Subsystem	Number of P.C. Boards	Failure Rate C/10^6 Hr.	Percentage of Total Failure Rate (%)
Central Processing Unit	9	152	25
Floating Point Processor	4	89	15
Cache Memory (1K by 16 bits)	4	66	11
Main Memory (64K by 16 bits)	4	296	49
Total	25	603	100

FAILURE RATE OF PDP-11/70 at 85°C

Subsystem	Number of P.C. Boards	Failure Rate (/10^6 Hr.)	Percentage of Total Failure Rate (%)
Central Processing Unit	9	383	4
Floating Point Processor	4	263	2
Cache Memory (1K by 16 bits)	4	199	2
Main Memory (64K by 16 bits)	4	9910	92
Total	25	10755	100

SUMMARY OF PERFORMANCE/COST ESTIMATES FOR THE MODULE, CHASSIS AND SYSTEM LEVEL BITS FOR A SINGLE PROCESSOR COMPUTER SYSTEM

Parameters	Module Level BIT	Chassis Level BIT	System Level BIT
P_{SFD}	65.4%	81.4% [1]	90% [1]
T_{SFD}	1-2 μsec	4.2 sec	2.44 msec
P_{FA}	0%	5.4%	0.15%
P_{LFE}	100%	100%	100%
T_{LFE}	1-2 μsec	4.2 sec	2.44 msec
A	21.5%	10.5%	[2]
P	16.9%	6.1%	[2]
FR	-13.3% [3]	5.4%	[2]
OS [4]	N.A.	N.A.	100
AS	N.A.	N.A.	N.A.
DS [4]	N.A.	783	1150

Notes:

[1] P_{SFD} at chassis and system level include the fault detection capability of the Module Level BIT.

[2] Cost of microdiagnostic hardware is included in the Module Level BIT for the CPU3 module.

[3] Negative failure rate increase is due to the use of error correcting code in the memory subsystem.

[4] Number indicate number of additional assembly language level instructions.

COMPUTER ARCHITECTURE/STANDARDIZATION/ COMMONALITY

Frank E. Ward

CENTACS

COMPUTER ARCHITECTURE/STANDARDIZATION/COMMONALITY

SESSION CHAIRPERSON: Frank E. Ward

CENTACS

SESSION SUMMARY

This session included three papers:

The first paper, by Harold Stone, U. of Mass., and A. H. Coleman of CENTACS, was on the subject: "Life Cycle Cost Study of the Effects of Standardization of Military Computer Architecture". This paper analyzed the relative life cycle cost of 78 Army/Navy Military Computer-based systems as a function of alternative instruction-set architecture scenarios.

The second paper, by William Burr of CENTACS, discussed the topic "A Bus System for the Military Computer Family". This paper discussed the results of a study to determine the requirements for military computer system buses or standard interfaces to permit meaningful life cycle competition and graceful technology insertion. Four buses were proposed: (1) an internal or I-bus interconnecting modules within a chassis, (2) an external or E-bus interconnecting chasis, (3) a low-speed peripheral bus and (4) a high-speed peripheral bus.

The third paper by William Dietz of Carnegie Mellon University was on the topic "Evaluation of Alternative Computer Architectures". In this paper a comparative evaluation of six alternative computer instruction-set architectures was made. A discussion of the methodology for evaluating instruction-set architectures was also presented.

Life-Cycle Cost Analysis of Instruction-Set Architecture Standardization for Military Computer-Based Systems

Harold S. Stone
University of Massachusetts

and

Aaron Coleman
U.S. Army CORADCOM

This life-cycle cost model reported here measured the effects of standardization of computer instruction-set architectures on military computer-based systems. The study considered six different scenarios, one of which assumes that standardization is not done, but only four different computer architectures are used across all systems. The remaining five scenarios considered the effects of standardizing on each of the architectures UYK-7, UYK-19, UYK-20, GYK-12, and UYK-41 (PDP-11).

Standardization impacts life-cycle costs in several ways. There is an inherent difference in the value and utility of the existing support software bases for the several architectures. The commercially supported architectures generate investment to augment and maintain a substantial portion of the software base free of government expenditure. Finally, some architectures are more efficient than others, and result in lower hardware costs if used as a standard.

The cost model attempts to incorporate these factors in a meaningful way to judge the relative importance of these factors and other factors on total life-cycle cost. The results of the model show that the CYK-41 results in the least life-cycle cost of any scenario over a broad range of annual rates of investment in support software. These conclusions are attributed to the fact that the CYK-41 ranks best or near best in each aspect that impacts total life-cycle cost.

I. Introduction

The objective of the study reported here is to measure the economic effects of standardization of computer instruction-set architectures on military computer-based systems. For the purposes of this study, the term architecture refers to the characteristics of a computer defined by its instruction repertoire. Two computers are said to have the same architecture if any assembly language program for one computer runs on the other and conversely. Two such computers may be vastly different in implementation and have radically different costs and performances. The commercial computer world has demonstrated that a family of implementations of a single architecture is feasible and desirable.

Military computer systems are starting to take on characteristics of commercial families in that prior-generation computers are being reimplemented with new hardware to take advantage of the technological improvements in cost and performance. At issue is the question of whether to standardize on a family of implementations of a single architecture or to use a mix of many architectures, each used in an environment best suited to it. If standardization appears to be attractive, then a further question is which computer architecture should be used as a standard. Standardization can realize potential savings by eliminating duplicate efforts, but on the other hand it can incur additional costs if a standard is used in an environment for which it is not well-suited. The question then becomes one of determining which standard is the best overall standard.

The method used in this study to suggest a best course of action is to compute the relative life-cycle cost for seventy-eight representative Army/Navy computer-based systems which are acquired and deployed over a twenty-two year interval. The systems are acquired in lots of twenty-six for each of three different time periods--1980, 1985, and 1990--with each lot deployed for ten years. R&D costs prior to each acquisition are included in the cost model. Thus the cost model serves to identify how a standard computer architecture can impact real life-cycle costs in terms of estimates of costs to be incurred, and gives some indication of the potential benefits and costs of the possible decisions. Any model of this type is subject to errors in estimates, so that the absolute dollar figures computed must be viewed as indicative of possible results rather than as predictions of the future. The model does identify the important factors, and estimates their relative importance if in fact the model does not predict dollar costs with absolute accuracy.

To isolate the key variable, computer architecture, from different hardware implementations, all computer systems are presumed to use the same family of modules and chassis in their implementation. These modules are presumed to be Military Computer Family (MCF) modules as specified by ITEK Corporation under contract to the U.S. Army.[1] The modules use a common collection of memory modules, input/output modules, and bus interfaces, with different CPU modules available to implement different instruction-set architectures. We presume that these modules can implement any of the instruction sets for the UYK-7, UYK-19, UYK-20, GYK-12, and UYK-41 (PDP-11) computers. The most likely methodology for realizing the collection of instruction sets from a common set of modules is to use CPU modules specific to each architecture that interface to the memory and input/output modules over a general bus.

The six scenarios studied are indicated in Table I.

The cost model identifies costs arising from the principal sources:

a. common costs, which are costs incurred to mount an architecture in the field apart from costs for hardware and applications software associated with each system that uses the architecture,

Table I

Scenario 1 Multi-architecture scenario. Field each computer system with the architecture most similar to its actual architecture. Select from choice of UYK-7, UYK-19, UYK-20, and GYK-12.

Scenario 2 UYK-7 Computer Family Architecture (CFA). Field each system with a UYK-7 architecture using a family member whose performance most nearly matches the requirements for that system.

Scenario 3 UYK-19 CFA. Similar to Scenario 2, but with a UYK-19.

Scenario 4 UYK-20 CFA. Similar to Scenario 2, but with a UYK-20.

Scenario 5 GYK-12 CFA. Similar to Scenario 2, but with a GYK-12.

Scenario 6 UYK-41 CFA. Similar to Scenario 2, but with a GYK-41.

b. hardware life-cycle costs, which include acquisition, logistics, and maintenance costs, and

c. software life-cycle costs, which include initial acquisition and operations and maintenance costs.

In this model, R&D costs are grouped with common costs, as are the costs for developing and maintaining software tools for an architecture.

Since the UYK-41 is solely a commercial development, there are dollars being expended on this architecture that do not have to be spent by the government if it is selected as a standard. To a lesser extent this is also true of the UYK-19 since it is based closely on the Data General NOVA computer. The other architectures considered are all military architectures essentially developed and owned by the government for which there exists little or no possibility of capturing commercial investment in the architecture. The model estimates the effects of capture of past and anticipated commercial investment in the life-cycle costs. These effects are incorporated into the common cost area.

Hardware life-cycle costs are impacted by architectural efficiency. Good computer architecture designs should tend to be efficient in the size of memory required to store programs and in the number of processor and memory cycles required to execute a program. An inefficient architecture may require more memory or a higher performance family member (or both) than an efficient architecture to do equivalent tasks. This study relies on estimates of architectural efficiency produced from statistical measurements made at Carnegie-Mellon University.[2] From the efficiency data, it is possible to construct computer configurations for each application and for each architecture where the configurations reflect the relative efficiency or inefficiency of the specific architecture. From the configuration data, we obtain the hardware costs for each scenario.

Software life-cycle costs as a function of architecture are most strongly impacted by the base of support software tools available for each architecture. Separate studies have surveyed the tools available[3] and estimated the effect of the different size bases on application software costs.[4] Together these studies provide a means for estimating software life-cycle costs in this report.

When all of the costs are considered together for the several scenarios, the UYK-41 architecture used as a family architecture results in the least life-cycle cost in our cost model. There are three primary contributing factors to this result. The model shows that architecture can impact life-cycle costs through

a. commercial investment in the architecture that can be captured by the government,

b. architectural efficiencies that result in reduced hardware costs, and

c. an extensive software tool base that leads to reduced applications software costs.

In each of these three categories, the UYK-41 is in the most favorable position, or very close to the most favorable. Our model shows that these three factors contribute to a cost savings of $1.5 billion (22 percent) of the cost of a UYK-19 CFA and about $5.1 billion (49%) of the cost of either a GYK-12 or UYK-7 CFA when the annual level of investment is held to $2 million. Regardless how the effects of architecture are modeled, because the UYK-41 is as good or better than the other architectures in each of the three key areas, the selection of a GYK-41 as a Computer Family Architecture should result in a reduction in the life-cycle costs when compared to the other alternatives treated here.

Section II of this report treats the details of the model. Section III contains the results of a sensitivity analysis that shows the results to be relatively insensitive to assumptions made here. The final section raises some points concerning weaknesses in the model and contains suggestions for future work to sharpen the model and to obtain more accurate estimates of true life-cycle costs.

II. The Life-Cycle Cost Model

There are six scenarios described in this report as cited in the introduction. Scenario 1, the Multi-Architectural Scenario, is intended to represent the state of affairs if no single standard architecture were selected and the Defense Department took reasonable steps to prevent further proliferation of instruction-set architectures for military applications. Thus all systems procured use one or more of the four prevalent military computer architectures UYK-7, UYK-19, UYK-20, and GYK-12, with each application using the actual architecture or nearest equivalent to the actual architecture. The remaining five scenarios treat the cases for which there is a single computer family architecture (CFA) adopted as a standard, and the architecture varies from scenario to scenario across the four military architectures and the UYK-41. To use a single architecture across a wide range of applications we assume that there exist at least three different implementations of an architecture in each family, namely, a microprocessor version, a minicomputer version, and a high-performance midi or maxi version.

For purposes of comparison, costs are measured as the life-cycle costs to acquire and deploy exemplary Army/Navy systems. The examples are shown in Table 2 according to the characteristics of configurations assumed to be fielded today. They are selected to be similar to actual systems deployed and in development today, and cover a wide variety of functions including airborne, naval surface, naval undersea, army tactical, and army command and control applications. The twenty-six systems listed in Table 2 is a sufficiently large cross section to be judged representative of the mix of systems to be procured in the next decade.

We recognize that the list of systems is incomplete, and does not represent systems as yet unconceived that may be developed during the time frame of the cost model. To account for the missing systems, we simply assume that additional systems similar to the ones in the table will be procured at later times, and that technological advances should be accounted in the cost of the hardware and increased programmer productivity. Thus, as part of the cost model we make the following assumptions:

a. the model accounts for the life-cycle cost of twenty-six systems with R&D beginning in 1978, hardware acquisition in 1980, programmer productivity averaged at 1983 levels, and the systems operated from 1980 until 1990.

TABLE 2 COMPUTER-BASED SYSTEM PARAMETERS (CONTD)

System	Architecture Assumed	Number of Processors	Total Memory Size	Program Size	Unused Memory	Number of I/O Ports
17	UYK-7*	1	64K x 32	32K x 32	4K	5
18	UYK-20	2	32K x 16	25K x 16	3K	3
	UYK-20*	1	32K x 16	25K x 16	3K	1
	UYK-20*	1	16K x 16	13K x 16	1K	1
	UYK-20*	1	8K x 16	6K x 16	0.5K	2
	UYK-20*	1	32K x 16	25K x 16	3K	2
19	GYK-12*	1	256K x 32	205K x 32	20K	4
20	UYK-7	2	64K x 32 (ea.)	50K x 32 (ea.)	2K (ea.)	2 (ea.)
21	UYK-7	2	176K x 32	140K x 32	18K	32
	UYK-7	2	144K x 32	115K x 32	15K	32
22	UYK-20	5	64K x 16 (ea.)	51K x 16 (ea.)	6K (ea.)	5 (ea.)
	UYK-7	3	576K x 32	460K x 32	20K	42
	UYK-7	2	384K x 32	305K x 32	15K	22
23	UYK-20	3	96K x 16 (ea.)	80K x 16 (ea.)	8K (ea.)	3
	UYK-7	1	192K x 32	160K x 32	15K	4
24	UYK-7	4	192K x 32 (ea.)	160K x 32 (ea.)	15K	8
	UYK-7	2	512K x 32	400K x 32	30K	16
25	UYK-20*	3	16K x 16 (ea.)	12K x 16 (ea.)	1K (ea.)	2 (ea.)
	UYK-20*	2	32K x 16	25K x 16	2K	4
	UYK-7	1	128K x 32	100K x 32	10K	6
	UYK-7	4	1056K x 32	800K x 32	50K	16
26	UYK-7	4	256K x 32	200K x 32	20K	30
	UYK-7	4	256K x 32	200K x 32	20K	30
	UYK-7	4	256K x 32	200K x 32	20K	30

*Architecture given in this table is approximately equivalent to, but not necessarily identical to, the actual architecture used.

b. to which is added the life-cycle costs of an identical set of systems procured in a batch five years later, and

c. to which is added the life-cycle costs of yet another identical batch of systems procured ten years after the first batch.

This brings the number of systems to a total of seventy-eight with twenty-six acquired in each of the years 1980, 1985, and 1990 and run for ten years. With R&D for the first batch beginning in 1978 and the last batch being retired in the year 2000, the model covers a time span of twenty-two years.

The cost equations for the model are summarized below.

The life-cycle cost for the acquisition of the computer systems in Scenario i is designated as C_i and is composed of three major components.

$$C_i = CC_i + \sum_j n_j H_{i,j} + \sum_j ASW_{i,j}$$

where

CC_i = common costs attributable to supporting the architecture(s) in Scenario i,

n_j = the number of copies of system j, j=1, 2, 3, ..., 26 to be acquired,

$H_{i,j}$ = the hardware life-cycle cost of one computer system j, j=1, 2, 3 26 as configured for Scenario i, and

$ASW_{i,j}$ = the life cycle cost of applications software for computer system j in Scenario i.

To compute total life cycle costs for each scenario, we sum the individual costs for common, hardware, and software costs. However, for both the hardware and software costs, we purchase each system three times, once in 1980, then again in 1985 and 1990. Details of the contributions to the individual terms appear in the remainder of this section. Results appear in the following section.

Common costs, C_i, are composed of three major components, and represent the costs incurred simply to have a fieldable computer architecture even before the first system is acquired and dep'oyed. These costs include such things as R&D costs, product planning, and software tools. So we write CC_i, the common costs for scenario i, as

$$CC_i = PP_i + HRD_i + SSW_i$$

where

PP_i = the product planning costs for scenario i

HRD_i = the costs for hardware ıesearch and development for scenario i, and

SSW_i = the costs associated with the support software research, development and support for scenario i.

Product planning costs, PP_i, cover architecture, hardware, and software expenditures that are incurred as part of the planning phase for the introduction of a new computer family. These cover such items as standardization and control of the specifications of the several aspects of the computer systems, and the identification of the functional characteristics for the low, middle, and high end of the family. The cost equation for product planning costs for Scenario i breaks down as

$$PP_i = PPA_i + PPH_i + PPS_i$$

where:

PPA_i = costs for product planning for architecture in Scenario i

PPH_i = costs for product planning for hardware in Scenario i, and

PPS_i = costs for product planning for support software in Scenario i.

Hardware R&D costs, HRD, cover expenditures for the R&D hardware expenditures required prior to fielding a system. For the scenarios considered, these costs are incurred as common costs and not tied to particular systems.

Support software costs, SSW_i, account for expenditures for software tools of general use to the computer community. Such tools as a DOD-I computer are included in these costs. SSW_i includes not only the initial acquisition of the software tools but maintenance and enhancement expenses as well. For tools in this category we assume that the annual maintenance and enhancement expenses are 0.3 times the acquisition costs.

It is crucial in the model to capture the effects of different sizes of software tool bases and the effects of commercial investment on costs incurred by the government in maintaining and acquiring tools. We also model the productivity of applications software programmers as a function of the tools available. Consequently we model the changing software tool base and the cost for maintaining the tools in the following manner.

We assume a fixed annual investment INV for SSW_i, and examine life-cycle costs as a function of this expenditure rate. Part of INV is used to maintain the existing base and the remainder is used to purchase new tools. As tools are added each year, the funds available for new purchases diminish as more and more of the funds are used to maintain existing software. We express the maintenance and logistics costs in year yr as $SSM_i(yr)$ given by the formula

$$SSM_i(yr) = .3\ GSB_i(yr)$$

where $GSB_i(yr)$ is the value of the government supported software base in year yr for scenario i.

To obtain the size of the software base in yr+1 given an annual investment of INV we have

$$\begin{aligned} GSB_i(yr + 1) &= GSB_i(yr) + (INV - .3\ GSB_i(yr)) \\ &= 0.7\ GSB_i(yr) + INV \end{aligned}$$

The term INV - 0.3 $GSB_i(yr)$ is the amount left over for new investment after

AD-A104 252 INTERNATIONAL BUSINESS SERVICES INC WASHINGTON DC F/G 9/2
US ARMY SOFTWARE SYMPOSIUM (2ND) HELD AT WILLIAMSBURG, VIRGINIA--ETC(U)
1978 S M TAYLOR DAAK70-78-D-0030

UNCLASSIFIED NL

9 OF [illegible]
AD
A104252

paying for operations and maintenance of the existing base. This formula is a well-known recursion for financial transactions, and can be solved to express GSB_i for any year in the future. Its solution is

$$GSB_i(yr + n) = .7^n\ GSB_i(yr) + INV\ (1 - (.7)^n)/.3$$

The total software base at any time in the future is the sum of the commercially supported base and the government supported base. Here we assume that the commercial base is fixed in time, and does not change, but the cost of the support of that base is borne totally by industry. Actually, the base will improve in time at no cost to the government. However, the government may have to pay license fees for the use of the software tools at software development centers that are ignored here, because they are likely to be negligible. The assumptions generally err in the favor of military architectures. The total software base in year yr + n is then described by the function:

$$TSB_i(yr + n) = .7^n\ GSB_i(yr) + INV\ (1 - (.7)^n)/.3 + CSB_i$$

where CSB_i is the value of commercially supported software base in Scenario i.

For the cost model, we have separately made a determination of the value of CSB_i and GSB_i (1978) for each scenario. These values and an assumed rate of investment INV yields TSB_i as a function of year during the time of the fielding of the software. The values of $TSB_i(1983)$, $TSB_i(1988)$, and $TSB_i(1993)$ are used to predict the cost of applications software.

Hardware life-cycle costs $H_{i,j}$ for system j in Scenario i break down into separate costs according to the equation

$$H_{i,j} = HA_{i,j} + HL_{i,j}$$

where

$HA_{i,j}$ = acquisition costs for one copy of system j in Scenario i, and

$HL_{i,j}$ = logistics costs for one copy of system j in Scenario i.

Separately we have determined that $HL_{i,j}$ average about 12% of $HA_{i,j}$ per year over a ten-year period, although expenditures are not actually uniform during that time. Thus for a ten-year life cycle, $HL_{i,j} = 1.2\ HA_{i,j}$, and

$$H_{i,j} = 2.2HA_{i,j}$$

To model the effects of improvement in hardware costs over time, we assume that hardware costs decrease by 20% per year on a declining balance basis. Thus if $HA_{i,j}(yr)$ is the hardware cost for system j in year yr, then

$$HA_{i,j}(yr + n) = HA_{i,j}(yr)(.8^n).$$

The 20% decrease per year has been the historical observation over the last decade for digital electronics as a whole, and is due largely to the very large number of units produced and the effects of product improvements and learning from the volume production methods.

For this study our procurement activities are staged in 1980, 1985, and 1990. We use the hardware costs

$$HA_{i,j}(1980) = .8^2\ HA_{i,j}(1978) = .64\ HA_{i,j}(1978)$$
$$HA_{i,j}(1985) = .8^7\ HA_{i,j}(1978) = .21\ HA_{i,j}(1978)$$
$$HA_{i,j}(1990) = .8^{12}\ HA_{i,j}(1978) = .069\ HA_{i,j}(1978)$$

Applications software life-cycle costs, $ASW_{i,j}$, like hardware life-cycle costs, break down into two terms

$$ASW_{i,j} = ASWA_{i,j} + ASWL_{i,j}$$

where

$ASWA_{i,j}$ = applications software acquisition costs for system j in Scenario i, and

$ASWL_{i,j}$ = applications software logistics costs for system j in Scenario i.

Again we estimate logistics costs to be proportional to acquisition costs. The estimate here is that $ASWL_{i,j}$ is about .45 $ASWA_{i,j}$ on an annual basis, or about 4.5 $ASWA_{i,j}$ for a ten year life-cycle. This coefficient is slightly higher than the .3 annual cost used for support software. Studies of actual data show this figure varies widely from system to system, and it is difficult to ascertain exactly what this figure should be. The factor 4.5 appears to be the right order of magnitude and may err in being too large rather than too small. Using 4.5 as the proportionality constant we obtain

$$ASW_{i,j} = ASWA_{i,j} + 4.5\ ASWA_{i,j} = 5.5\ ASWA_{i,j}.$$

There remains to relate the size of the support software base to the cost of acquisition of applications software.

To get at this last cost we make use of data developed by W. Svirsky, T. Giles, and A. Irwin of System Development Corporation.[5] Figure 1 shows a set of curves of programmer productivity as a function of software tools available. The X-axis shows increasing tools as a fraction of an idealized full software base. The Y-axis shows the cost per machine language instruction. Intuitively, as more tools are brought into play the cost per line of code diminishes. The sources of this data were SDC project managers for five large-scale command and control software systems. Raw data included the number of instructions generated, the expected percentage increase in project cost that would result if each of thirteen software tools available to them had not been used, and the expected decrease in project cost that would result if an ideal

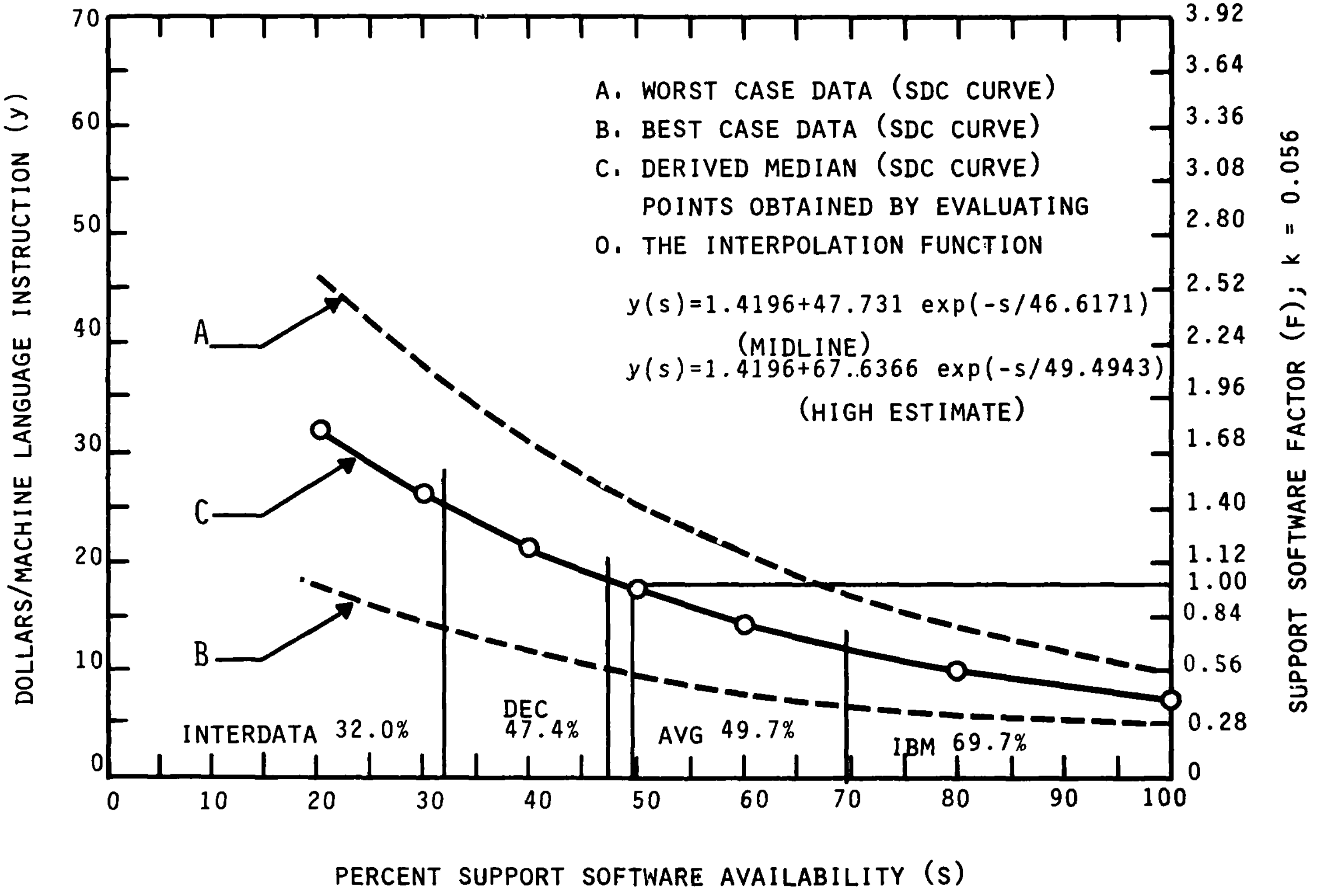

70
60
50
40
30
20
10
0
DOLLARS/MACHINE LANGUAGE INSTRUCTION (y)
0
10
20
30
40
50
60
70
80
90
100
PERCENT SUPPORT SOFTWARE AVAILABILITY (S)
3.92
3.64
3.36
3.08
2.80
2.52
2.24
1.96
1.68
1.40
1.12
1.00
0.84
0.56
0.28
0
SUPPORT SOFTWARE FACTOR (F); k = 0.056
A. WORST CASE DATA (SDC CURVE)
B. BEST CASE DATA (SDC CURVE)
C. DERIVED MEDIAN (SDC CURVE)
POINTS OBTAINED BY EVALUATING
O. THE INTERPOLATION FUNCTION
y(s)=1.4196+47.731 exp(-s/46.6171)
(MIDLINE)
y(s)=1.4196+67.6366 exp(-s/49.4943)
(HIGH ESTIMATE)
A
C
B
INTERDATA 32.0%
DEC 47.4%
AVG 49.7%
IBM 69.7%

FIGURE 1

software tool base were available. The median curve represents the best guess as to the actual cost function but the guess is largely judgmental. We have found that the upper curve is more representative of actual costs incurred in 1978, so we use it for this study. Again there is some caution to be observed in using this data because true productivity depends on many factors besides tools available. The costs per line of code are computed as follows. Let

$$s_i = (\mathrm{TSB}_i/\mathrm{IDEAL}) \times 100$$

where TSB_i is the total software base value, and IDEAL is the value of the ideal base. Then

$$\mathrm{PROD}_i = 1.4196 + 47.731 \exp(-s_i/46.6171)$$

where PROD_i is the cost per line of code in Scenario i. Finally,

$$\mathrm{ASWA}_{i,j} = m_j \mathrm{Prod}_i$$

where m_j is the total number of lines of code for system j. Since software is procured at three different times in this model, the value of TSB_i is the value of the base at the time of the procurement.

Summary of variables and equations

C_i = life-cycle cost for scenario i.

CC_i = common costs for scenario i.

$H_{i,j}$ = hardware life cycle costs for system j in scenario i.

n_j = the number of copies of system j acquired

$ASW_{i,j}$= the applications software life-cycle cost for system j in scenario i

PP_i = product planning costs for scenario i.

PPA_i = product planning costs for architecture in scenario i.

PPH_i = product planning costs for hardware in scenario i.

PPS_i = product planning costs for software in scenario i.

HRD_i = hardware R&D costs for scenario i

SSW_i = support software costs for scenario i

INV = annual rate of investment in support software

$GSB_i(yr)$ = government supported software base in year yr in scenario i

$TSB_i(yr)$ = total software base in year yr of scenario i

CSB_i = commercially supported software base in scenario i

$HA_{i,j}$ = hardware acquisition costs for system j in scenario i

$HL_{i,j}$ = hardware logistics costs for system j in scenario i

$ASWA_{i,j}$ = applications software acquisition costs for system j in scenario i

$ASWL_{i,j}$ = applications software logistics costs for system j in scenario i

IDEAL = value of ideal software base

s_i = percentage value of total software base with respect to ideal base in scenario i

m_j = number of machine language instructions for system j

$PROD_i$ = cost per line of machine language instruction in scenario i

$$C_i = CC_i + \sum_j n_j \; H_{i,j} + \sum_j ASW_{i,j}$$

$$CC_i = PP_i + HRD_i + SSW_i$$

$$PP_i = PPA_i + PPH_i + PPS_i$$

$$SSW_i = 22 \cdot INV$$

$$GSB_i(yr + n) = .7^n \; GSB_i(yr) + INV \; (1 - (.7)^n)/.3$$

$$TSB_i(yr) = GSB_i(yr) + CSB_i$$

$$H_{i,j} = HA_{i,j} + HL_{i,j} = 2.2 \; HA_{i,j} = 2.2(HA_{i,j}(1980) + HA_{i,j}(1985) + HA_{i,j}(1990))$$

$$HA_{i,j}(yr + n) = (.8)^n \; HA_{i,j}(ye)$$

$$ASW_{i,j} = ASWA_{i,j} + ASWL_{i,j} = 5.5\ ASWA_{i,j} = 5.5\ (ASWA_{i,j}(1983) + ASWA_{i,j}(1988) + ASWA_{i,j}(1993))$$

$$s_i(yr) = TSB_i(yr)/IDEAL$$

$$PROD_i(yr) = 1.4196 + 47.731\ \exp(-s_i(yr)/46.6171)$$

$$ASWA_{i,j}(yr) = m_j\ PROD_i(yr)$$

III. Summary of independent input data and results of computations

The life-cycle costs as a function of annual investment in support software tools is given in Fig. 2. Observe that the multiarchitecture scenario is considerably greater than the other scenarios in life-cycle cost because it accounts for maintaining common costs at a much higher level than the scenarios with a standard computer architecture while achieving poorer results on improving productivity of applications programmers since tools development is spread over four architectures. Actually, the situation is probably much worse today than is modeled by the multi-architecture scenario since many more than four architectures are fielded as of today.

The UYK-41 life-cycle cost lies below the curves for all levels of investment shown. At higher levels of investment, the curves tend to turn upward as the cost of new tools is not returned by increased programmer productivity. Consequently, it is doubtful that the investment dollars will be maintained at that level. The independent data inputs that account for the shape of the curves appear in the remainder of this section.

Common cost data are summarized in tables 3 and 4 for product planning and R&D costs, respectively. Briefly the assumptions of a level of effort are based on the level required for a single standard as estimated by the effort now expended by commercial and military organizations for similar functions. This comes to five man-years per year for each of the architecture, hardware, and software product planning efforts. For the commercial architectures UYK-19 and

FIGURE 2
TOTAL LIFE CYCLE COSTS VS ANNUAL SUPPORT SOFTWARE EXPENDITURES

ALL PARAMETERS AT NOMINAL VALUES

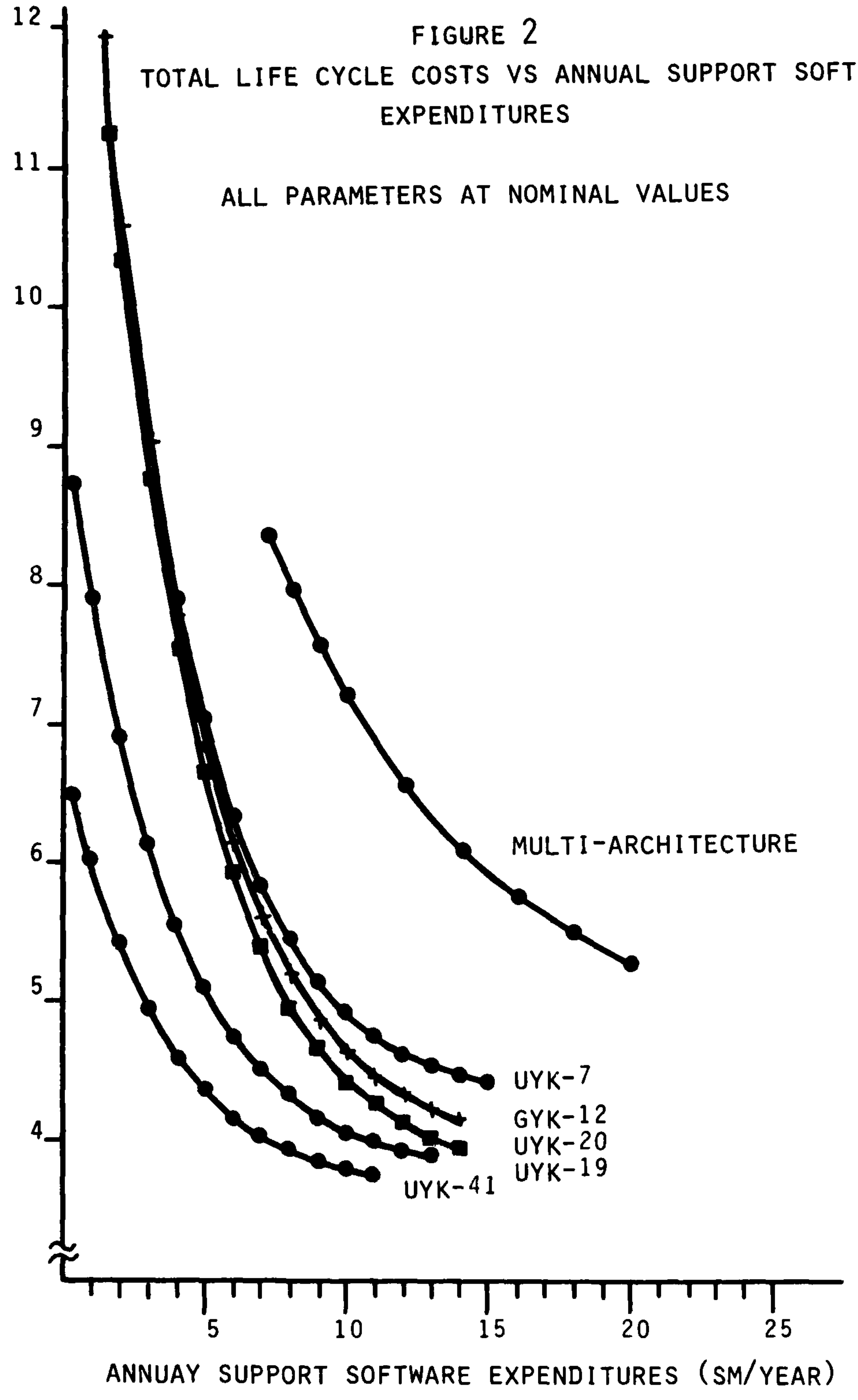


TABLE 2 COMPUTER-BASED SYSTEM PARAMETERS

System	Architecture Assumed	Number of Processors	Total Memory Size	Program Size	Unused Memory	Number of I/O Ports
1	UYK-19	1	64K x 16	40K x 16	1K	7
2	UYK-20	1	64K x 16	20K x 16	32K	4
3	UYK-19	1	24K x 16	22K x 16	1K	7
4						
	UYK-7*	3	160K x 32	71K x 32	22K	5
	UYK-7*	3	96K x 32	76K x 32	2K	5
5	UYK-19*	1	2K x 8	1K x 8	0	2
6	GYK-12	2	131K x 32	68K x 32	26K	4
7	UYK-19*	1	32K x 16	24K x 16	3K	4
8						
	UYK-7*	1	131K x 32	102K x 32	5K	7
	UYK-20*	3	64K x 16 (ea.)	51K x 32 (ea.)	3K (ea.)	
9	UYK-19*	1	18K x 16	1K x 16	2K	2
10						
	GYK-12	1	512K x 32	223K x 32	10K	10
	GYK-12	1	390K x 32	141K x 32	7K	10
11	GYK-12	2	64K x 32	24K x 32	5K	12
12	GYK-12	1	64K x 32	62K x 32	2K	16
13	UYK-19*	1	72K x 16	48K x 16	2K	17
14						
	UYK-20	2	128K x 16	64K x 16	19K	16
	UYK-7	1	48K x 32	19K x 32	7K	2
15						
	UYK-20*	1	64K x 16	4K x 16	2K	5
	UYK-20*	1	32K x 16	26K x 16	1K	2
16						
	UYK-19	2	64K x 16 (ea.)	18K x 16 (ea.)	30K (ea.)	5
	UYK-19	1	64K x 16	18K x 16	5K	2

*Architecture given in this table is approximately equivalent to, but not necessarily identical to, the actual architecture used.

Table 3

Product Planning Life-Cycle Costs, 1978-1987

Scenario	Annual PPA_i	Effort funded by Government	Annual PPH_i	Effort funded by Government	Annual PPS_i	Effort funded by Government	Total funded, 10-years, Government
1 Multiple Families	20 MY	18.5 MY	10 MY	10 MY	10 MY	10 MY	385 MY
2 UYK-7 CFA	5 MY	5 MY	5 MY	5 MY	5 MY	5 MY	150 MY
3 UYK-19 CFA	5 MY	3.5 MY	5 MY	5 MY	3.5 MY	3.5 MY	120 MY
4 UYK-20 CFA	5 MY	5 MY	5 MY	5 MY	5 MY	5 MY	150 MY
5 GYK-12 CFA	5 MY	5 MY	5 MY	5 MY	5 MY	5 MY	150 MY
6 UYK-41 CFA	5 MY	2.5 MY	5 MY	5 MY	5 MY	2.5 MY	100 MY

1 MY = 1 Man-year charged at $75,000 per man year, 1978 dollars

UYK-41, some of the architecture and software costs for the government are reduced because of the ability to capture commercial investment. For hardware product planning no commercial investment is captured since military hardware is a good deal different from commercial hardware. The multi-architecture scenario is charged a small multiple of the costs for mounting a single architecture because of the required duplication of effort.

The last term in the common cost equation is SSW_i, the support software investment. This we set as an independent variable equal to 22 times the annual rate of investment INV in creating the curves in Fig. 2. An estimate of realistic values of INV based on present and past experience is that INV is likely to fall into the \$2 million to \$4 million per year region, and unlikely to reach \$10 million levels unless the funding methodology for such programs is changed considerably by DOD. Our models suggest that even at \$10 million per year investment there is a positive return on the investment dollars. However, the dollar values are not discounted annually. With discounting taken into account, return is somewhat lower than indicated here.

Hardware costs are developed from reasonably accurate estimates of MCF module costs based on the cost of similar modules today. The basic costs appear in Table 5. Note that commercial architectures are charged a 2% royalty on the processor modules to account for royalties that may well have to be paid. Each of the twenty-six systems was configured from MCF modules, and the details of the configurations appear in the full cost-model report.[6] The configurations take into account architectural efficiency as measured separately at Carnegie-Mellon University.[2] The raw efficiency measures used in this study appear in Table 6a. These however are interim results of the measurement process. Final data from the measurements appear in Table 6b, but were computed

TABLE 5 MCF MODULE PRODUCTION COSTS (1977 $)

Module	Description	Cost
CPU-1	UYK-41 CPU, LSI-11 equivalent	3.6K*
CPU-2	UYK-41 CPU, PDP-11/34 equivalent	10.2K*
CPU-3	UYK-41 CPU, PDP-11/70 equivalent	25.5K*
CPU-6	UYK-7 CPU, equivalent to current model	30 K
CPU-6S	UYK-7 CPU, slow-speed version	20 K
CPU-6M	UYK-7 CPU, microprocessor version	6 K
CPU-7	GYK-12 CPU, equivalent to current model	25 K
CPU-7S	GYK-12 CPU, slow-speed version	10 K
CPU-7M	GYK-12 CPU, microprocessor version	6 K
CPU-8	UYK-19 CPU, equivalent to current model	10.2K*
CPU-8F	UYK-19 CPU, high-speed version	25.5K*
CPU-8M	UYK-19 CPU, microprocessor version	3.6K*
CPU-9	UYK-20 CPU, equivalent to current model	10 K
CPU-9F	UYK-20 CPU, high-speed version	25 K
CPU-9M	UYK-20 CPU, microprocessor version	3.5K
NRAM16	Memory, nonvolatile, 16K x 16 bit	8 K
NRAM32	Memory, nonvolatile, 32K x 16 bit	10 K
NRAM64	Memory, nonvolatile, 64K x 16 bit	13 K
PROM32	Memory, read-only, 32K x 16 bit	10 K
MCM-1	Memory controller	5 K
MCM-2	Memory controller	5 K
MCM-3	Memory controller	5 K
IOP-1	I/O Processor, UYK-20 Systems	5 K
IOP-2	I/O Processor, GYK-12 Systems	10 K
IOP-3	I/O Processor, UYK-7 Systems	15 K
IOX	I/O Exchange, 8-way, GYK-12 Systems	5 K
BEM-1	Bus extender module	5 K
NIM FAST	NTDS FAST interface	2.5K
NIM SLOW	NTDS SLOW interface	2.5K
RIM	RS-232 interface	2.5K
DIM	Discrete interface	2.5K
DMAC	Direct memory-access controller	5 K
PCM-1	Power converter	10 K
PCM-2	Power converter, extra 5V capacity	12 K
FULL ATR	ATR Chassis, includes $2K for assembly/checkout	12 K
SHORT ATR	ATR Chassis, includes $2K for asmbly/ck	11 K
HALF ATR	ATR Chassis, includes $2K for asmbly/ck	9 K
	Systems integration for multiple chassis	2 K

*Includes estimated 2% royalty for CPU modules

TABLE 6 ARCHITECTURE COMPARISONS

A. ARCHITECTURE EFFICIENCY MEASUREMENTS, INTERIM DATA

ARCHITECTURE	S-MEASURE	M-MEASURE
UYK-7	1.24	1.38
UYK-19	92	1.18
UYK-20	89	.73
GYK-12	1.12	.96
GYO-21	.82	.88

B. ARCHITECTURE EFFICIENCY MEASUREMENTS, FINAL DATA

HITECTURE	S-MEASURE	M-MEASURE	R-MEASURE
UYK-7	1.30	1.38	1.12
UYK-19	.93	1.18	1.17
UYK-20	.89	.73	.77
GYK-12	1.14	.96	.96
GYQ-21	.82	.88	1.03

Storage Utilization Efficiency
Memory Activity Efficiency
Processor Activity Efficiency

too late to be used in the configurations from which the cost model was computed.

The S-measure is a measure of the storage utilization efficiency for programs. If Architecture A has a storage efficiency measure say 20% higher than Architecture B, then A requires approximately 20% more storage to hold programs than A does. Configurations used a number of memory modules to contain the system data plus a number computed to be sufficient to hold the program for each application. The latter number varied from architecture to architecture according to the S-Measure.

The M-measure is a measure of storage reference efficiency. If Architecture A has an M-measure say 20% higher than Architecture B, then it must go to storage about 20% more frequently than Architecture B to do the equivalent task. This is reflected in the configurations by the selection of family member for each system. Inefficient architectures tend to require higher performance family members than efficient architectures. The substitution chart for family members is shown in Table 7.

The R-measure is a measure of processor cycle efficiency. Here, one architecture may require a faster internal clock than another to do an equivalent task. This was developed too late to be used in the configurations but the effects of the R-measure are rather small when compared to total life-cycle costs.

Hardware acquisitions costs by system appear in Table 8, and total life-cycle costs for hardware appear in Table 9.

This brings us to applications software life-cycle costs. A survey of the available software for the several architectures was conducted during the course of this study and for a prior study.[3,8] The survey investigated the availability of roughly thirty individual software tools from such sources as

TABLE 7 RELATIVE PERFORMANCE DATA

A. EQUIVALENT GYQ-21 PERFORMANCE

Architecture	Est. KOPS	Normalization Factor	Adjusted KOPS
GYQ-21 (2)	500	1 x .88 / .88	500
GYQ-21 (3)	800	1 x .88 / .88	800
UYK-7	500	2 x .88 / 1.38	640
UYK-19	500	1 x .88 / 1.18	375
UYK-20	400	1 x .88 / .73	480
GYK-12	400	2 x .88 / .96	736

B. PROCESSOR REPLACEMENT CHART

	REPLACE SPECIFIED CPU WITH:				
Architecture Specified	Scenario 2 UYK-7 CFA	Scenario 3 UYK-19 CFA	Scenario 4 UYK-20 CFA	Scenario 5 GYK-12 CFA	Scenario 6 GYQ-21 CFA
UYK-7	CPU-6	CPU-8F	CPU-9F	CPU-7	CPU-3
UYK-7 (micro)	CPU-6M	CPU-8M	CPU-9M	CPU-7M	CPU-1
UYK-19	CPU-6S	CPU-8	CPU-9	CPU-7S	CPU-2
UYK-19(micro)	CPU-6M	CPU-8M	CPU-9M	CPU-7M	CPU-1
UYK-20	CPU-6	CPU-8F	CPU-9	CPU-7S	CPU-2
UYK-20(micro)	CPU-6M	CPU-8M	CPU-9M	CPU-7M	CPU-1
GYK-12	CPU-6	CPU-8F	CPU-9F	CPU-7	CPU-3
GYK-12(micro)	CPU-6M	CPU-8M	CPU-9M	CPU-7M	CPU-1

TABLE . HARDWARE ACQUISITION COSTS $n_j HA_{i,j}$ BY SCENARIO

System	n_j	Scenario 1 Multi-arch.	Scenario 2 UYK-7 CFA	Scenario 3 UYK-19 CFA	Scenario 4 UYK-20 CFA	Scenario 5 GYK-12 CFA	Scenario 6 GYQ-21 CFA
1	50	4,735	5,525	4,735	4,725	5,025	4,735
2	192	17,184	22,944	20,160	17,184	21,024	17,222.4
3	100	9,070	10,050	9,070	9,050	9,050	9,070
4	150	137,700	137,700	131,250	130,800	135,450	126,150
5	3325	118,370	142,975	118,370	118,037.5	152,950	118,370
6	616	179,872	186,032	180,488	179,872	179,872	180,488
7	9	748.8	873	748.8	747	801	748.8
8	8	3,524	4,004	3,420	3,404	3,484	3,332.8
9	800	30,080	32,000	30,080	30,000	32,000	30,080
10	28 Div	10,584	11,452	9,870	9,856	10,584	9,506
	94 Bat	29,046	31,960	27,871	27,824	29,046	26,649
11	27	5,292	6,102	5,319	5,292	5,292	5,049
12	30	4,680	4,830	4,695	4,680	4,680	4,695
13	832	138,278.4	146,432	138,278.4	138,112	138,112	138,278.4
14	64	20,576	21,856	22,272	20,256	20,896	19,014.4
15	300	23,250	24,750	23,310	23,250	24,750	23,310
16	30	7,593	8,475	7,593	7,575	7,575	7,593
17	16	1,768	1,768	1,696	1,688	1,688	1,696
18	800	267,600	394,000	293,040	267,600	298,000	268,560
19	200	42,000	43,000	36,900	36,800	42,000	36,900
20	190	73,340	73,340	71,630	71,440	71,440	71,630
21	10	10,480	10,480	10,020	10,000	10,280	9,460
22	70	137,305	158,305	125,986	124,215	126,455	125,300
23	27	14,080.5	16,510.5	13,446	13,405.5	14,890.5	13,165.2
24	90	190,620	190,620	175,590	174,420	182,520	170,190
25	12	22,356	23,676	20,785.2	19,992	22,056	19,554
26	26	61,152	61,152	57,408	57,252	59,592	56,628
Totals		1,561,285	1,770,811.5	1,544,031	1,507,477	1,609,442.5	1,497,375

All figures are in thousands of dollars

TABLE [illegible] HARDWARE LIFE CYCLE COSTS

$$\sum_j n_j H_{i,j}$$

Year	Scenario 1 Multi-arch.	Scenario 2 UYK-7 CFA	Scenario 3 UYK-19 CFA	Scenario 4 UYK-20 CFA	Scenario 5 GYK-12 CFA	Scenario 6 GYQ-21 CFA
1978	3,435M	3,896M	3,397M	3,316M	3,541M	3,294M
1980 -(.64 of 1978)	2,198M	2,493M	2,174M	2,123M	2,266M	2,108M
1985 (.21 of 1978)	720M	817M	712M	696M	743M	691M
1990 (.069 of 1978)	236M	268M	233M	228M	243M	226M
Total: 1980-90	3,155M	3,578M	3,120M	3,046M	3,252M	3,026M

1 M = $1 Million

the industrial manufacturer of the architecture, the government, or private sources that offer the software as a product. The value of the software bases for this study and percentage of the ideal software base value of $24 million appears in Table 10, together with an estimate of the initial cost per machine language instruction for application programs using each architecture in 1978 on typical military computer systems. Table 11 shows how the software costs vary across the twenty-six systems in 1978. The functions that show how productivity changes as the software changes were used to compute productivity in 1983, 1988, and 1993 in each scenario.

IV. Sensitivity analysis

The model presented contains a number of assumptions and mathematical models for which there is insufficient empirical data to be sure of the correctness of the assumptions. To test the results produced by the model it is essential to see if the results are sensitive to perturbations in the assumptions.

Among the major areas in question are

1. the coefficient .45 that represents annual applications software logistics costs as a fraction of procurement costs,
2. The coefficient .3 that represents annual support software logistics costs as a fraction of procurement costs
3. a 20% per year annual decrease in hardware costs
4. productivity increases dependent on the software tool base, and
5. reduced government investment expenditures for the commercial architectures.

Figures 3, 4, and 5 show the results of perturbing the model in different ways. Figure 3 shows the effects of software costs doubling from the modeled value, where the doubled costs might be due to a change in any assumption or combination of assumptions that impact software cost. These curves generally

TABLE 10 PROGRAMMER PRODUCTIVITY

1978 BASE

Architecture	Available Base	% of Req'd. Base	Est. Cost Per Instr.
UYK-7	13,325M	55.8	$23.34
UYK-19	9,665M	40.4	$31.32
UYK-20	5,105M	21.4	$45.31
GYK-12	4,445M	18.6	$47.87
GYQ-21	14,865M	61.6	$20.90

TABLE 11 SCENARIO 1

APPLICATIONS SOFTWARE ACQUISITION COSTS, 1978

System	Architecture	Instruction Count	ASWA Acquisition Cost	ASW = 5.5 x ASWA
1	UYK-19	26.7K	.8M	4.6M
2	UYK-20	13.3K	.6M	3.3M
3	UYK-19	14.7K	.5M	2.5M
4	UYK-7	142 K	3.3M	18.2M
5	UYK-19	.3K	.01M	.05M
6	GYK-12	91.0K	4.4M	24.0M
7	UYK-19	16.0K	.5M	2.8M
8				
	UYK-7	100 K	2.3M	12.8M
	UYK-20	85.3K	3.9M	21.3M
9	UYK-19	.7K	.02M	.1M
10	GYK-12	485 K	23.2M	127.7M
11	GYK-12	32 K	1.5M	8.4M
12	GYK-12	85.3K	4.1M	22.5M
13	UYK-19	32.0K	1.0M	5.5M
14				
	UYK-20	42.7K	1.9M	10.6M
	UYK-7	25.3K	.6M	3.2M
15	UYK-20	21.9K	1.0M	5.5M
16	UYK-19	34.7K	1.1M	6.0M
17	UYK-7	42.7K	1.0M	5.5M
18	UYK-20	344 K	15.6M	85.7M
19	GYK-12	273 K	12.4M	68.0M
20	UYK-7	171 K	4.0M	22.0M
21	UYK-7	341 K	8.0M	43.8M
22				
	UYK-20	170 K	7.7M	42.4M
	UYK-7	2,048 K	47.8M	262.9M
23				
	UYK-20	153 K	6.9M	38.1M
	UYK-7	410 K	9.6M	52.6M
24	UYK-7	2,047 K	47.8M	262.3M
25				
	UYK-20	64 K	2.9M	15.9M
	UYK-7	1,894 K	44.2M	243.1M
26	UYK-7	1,640 K	38.3M	210.5M
Totals		10,847 K	$297.5M	$1,636.3M

FIGURE 3

TOTAL LIFE-CYCLE COSTS VS ANNUAL SUPPORT SOFTWARE EXPENDITURES

APPLICATIONS SOFTWARE COSTS AT 2 X NUMINAL VALUE

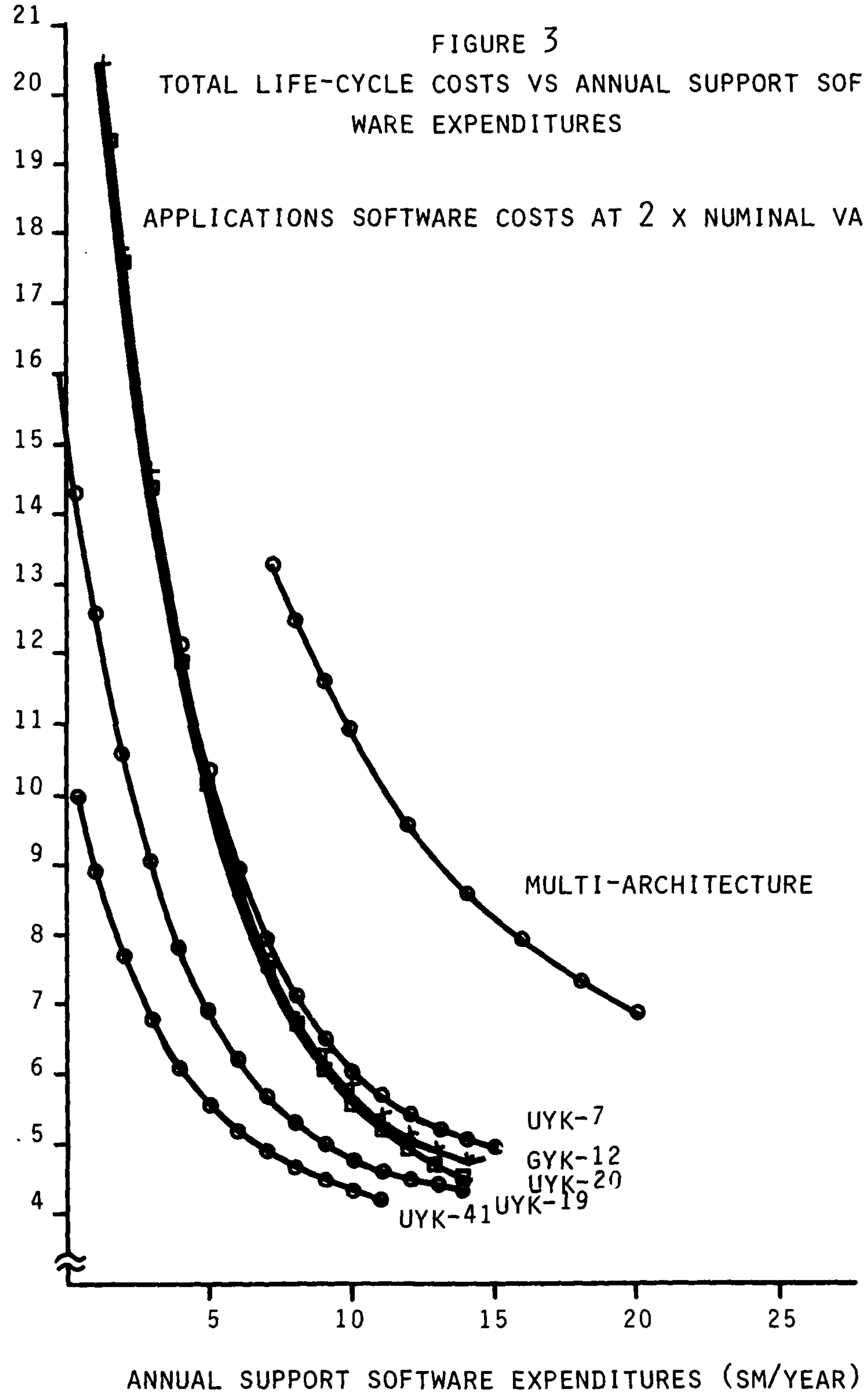


FIGURE 4

TOTAL LIFE-CYCLE COSTS VS ANNUAL SUPPORT SOFTWARE EXPENDITURES

APPLICATIONS SOFTWARE COSTS AT .5 X NOMINAL VALUE

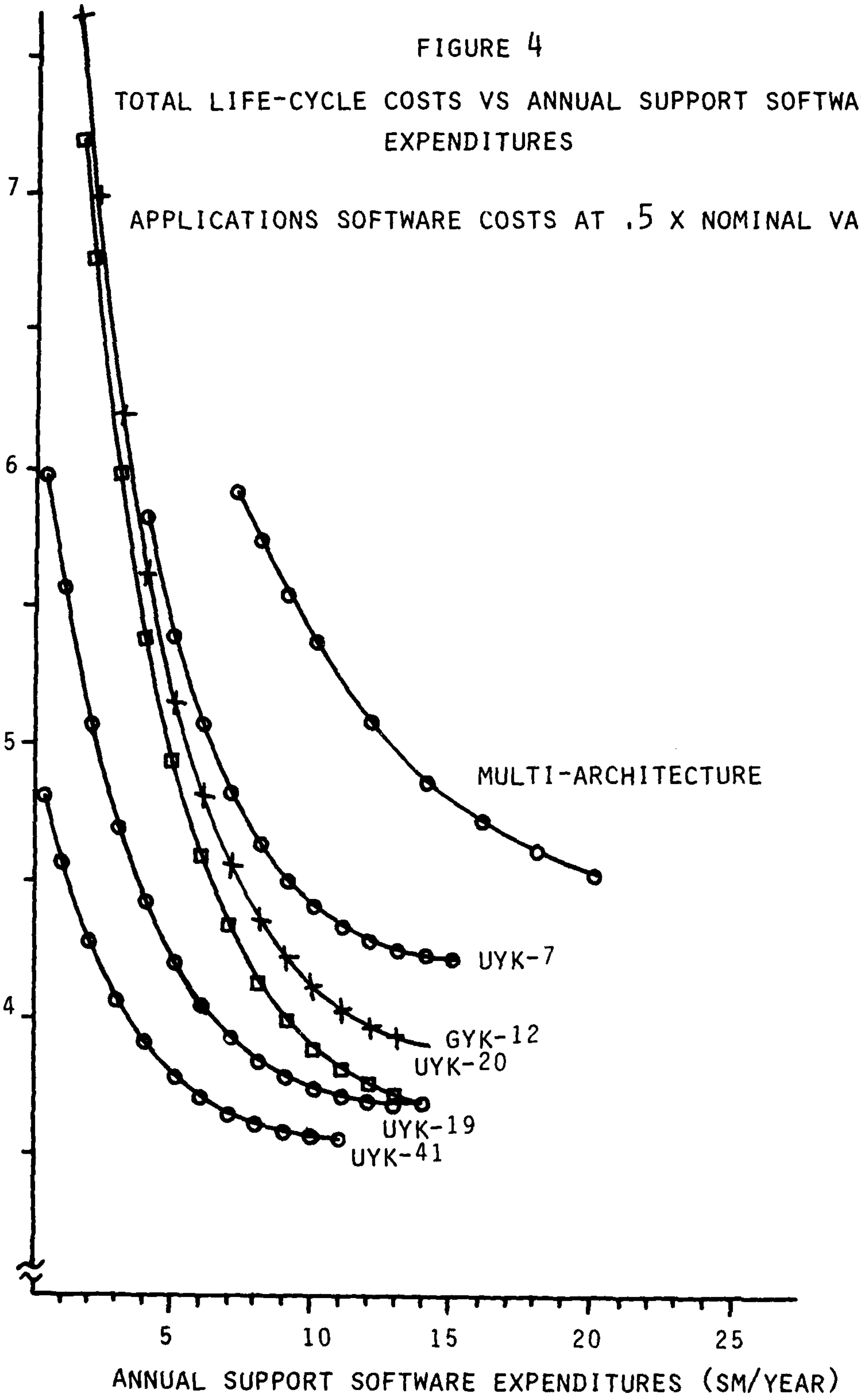


FIGURE 5

TOTAL LIFE-CYCLE COSTS VS ANNUAL SUPPORT SOFTWARE EXPENDITURES

ANNUAL SUPPORT SOFTWARE LOGISTICS EXPENSES = 10% OF SOFTWARE BASE VALUE

(REDUCED FROM 30 PERCENT OF SOFTWARE BASE VALUE)

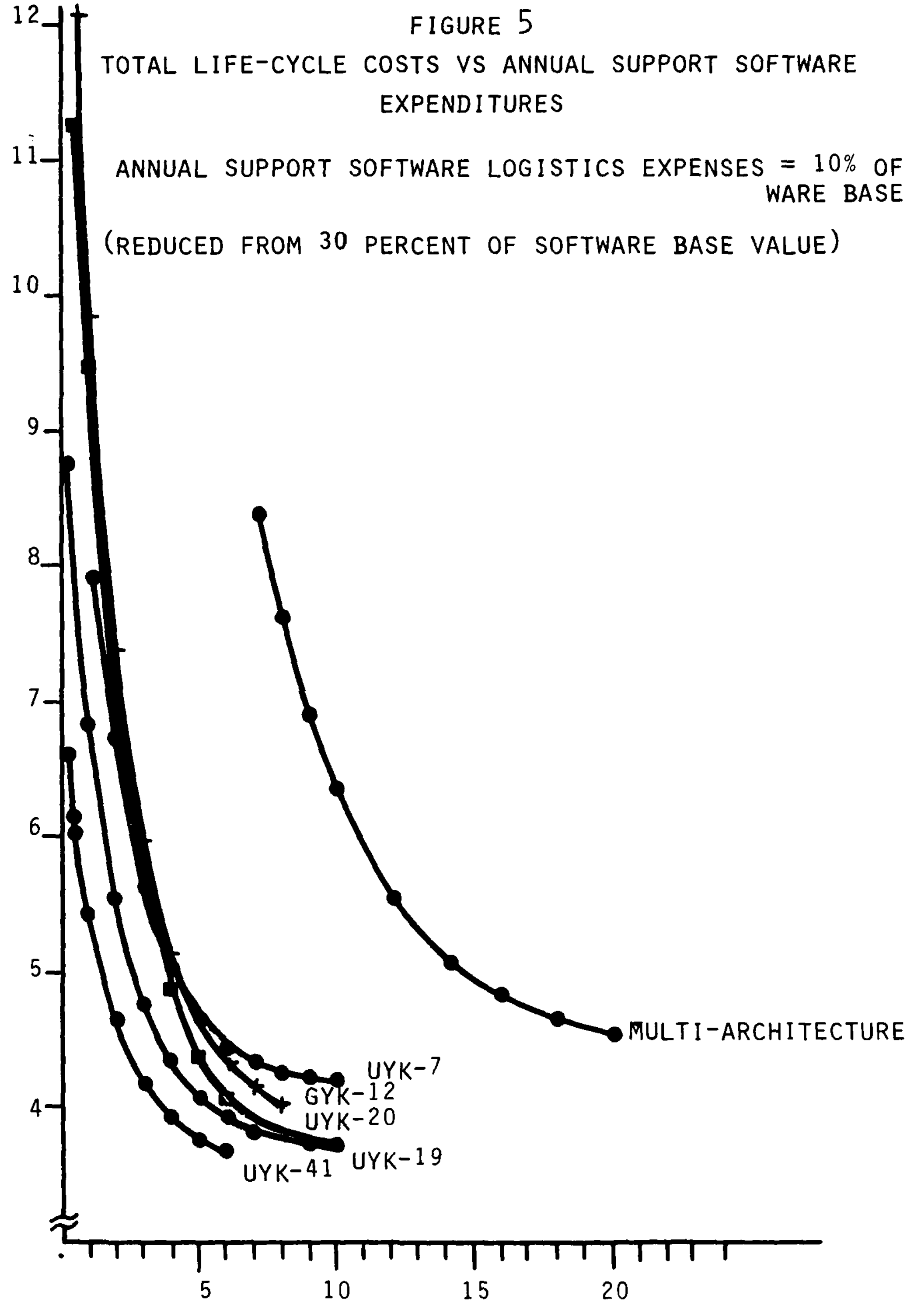


Figure 6 TOTAL LIFE-CYCLE COSTS VS ANNUAL SUPPORT SOFTWARE EXPENDITURES

Annual Support Software Logistics Expenses = 10 percent of Software Base Value

(Reduced from 30 percent of Software Base Value)

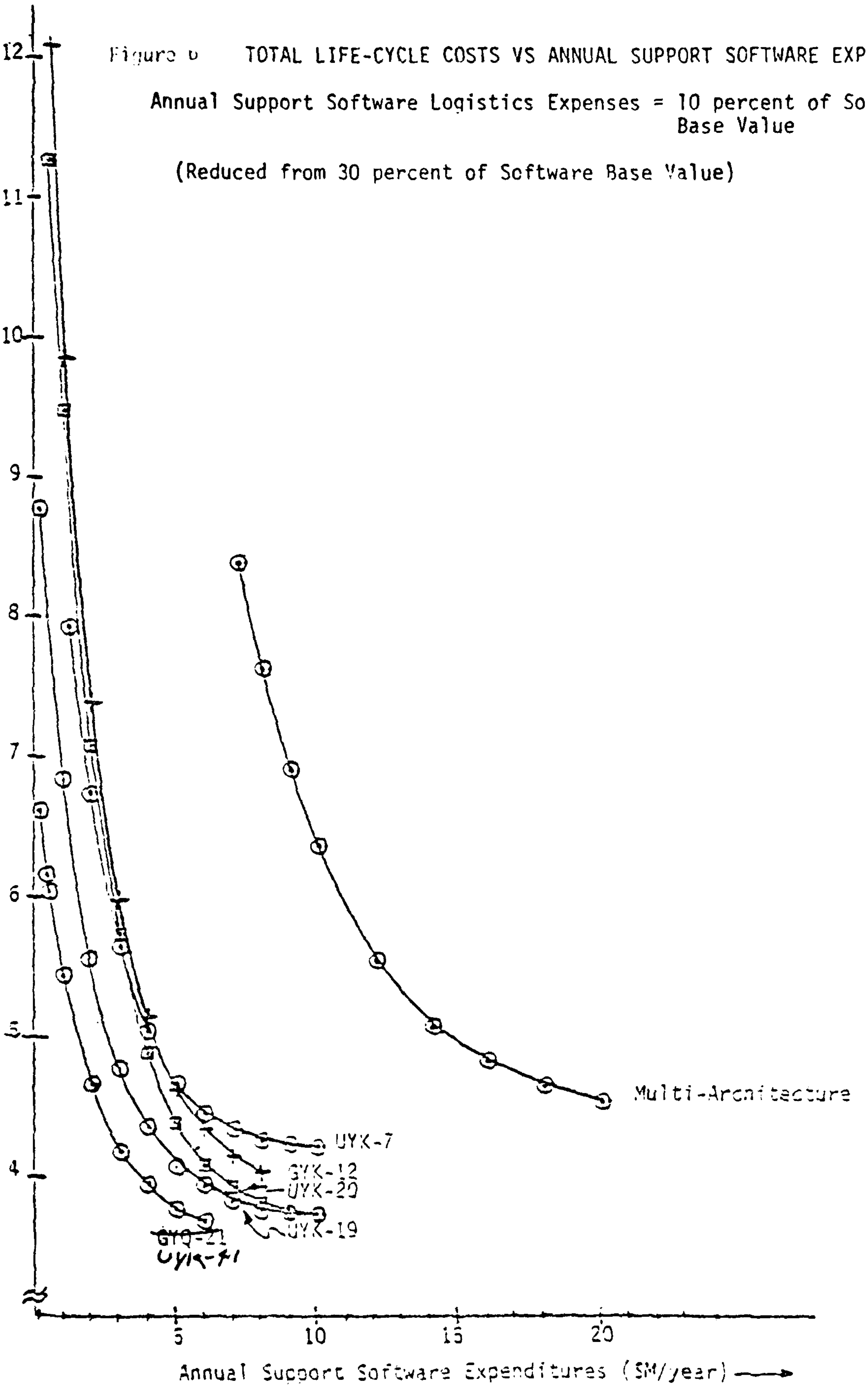


FIGURE 6. HRD_J, HARDWARE R&D LIFE-CYCLE COSTS

SCENARIO	HRD_J/YR	HRD_J, 12 YEAR LIFE-CYCLE COSTS
1 - MULTIPLE FAMILY	$10M	$130M
2 - UYK-7 CFA	$ 5M	$ 60M
3 - UYK-19 CFA	$ 5M	$ 60M
4 - UYK-20 CFA	$ 5M	$ 60M
5 - GYK-12	$ 5M	$ 60M
6 - GYQ-21 CFA	$ 5M	$ 60M

$1M = $1 MILLION

CENTACS

show that the UYK-7, GYK-12, and UYK-20 scenarios lead to approximately equal life cycle costs, with the UYK-19 and UYK-41 benefiting from commercial investment in support software. For sufficiently high levels of investment the UYK-20 results in lower costs than both the GYK-12 and UYK-7 as the effects of its greater hardware efficiency become evident.

Figure 4 shows the effects of much greater productivity in the software area than at the nominal value of the model. With software costs much lower, the differences are accounted principally from differences in hardware efficiency and commercial investment. Now for sufficiently high levels of investment in software, the UYK-7 advantage of a large software tool base becomes less important and it yields a higher life-cycle cost as a standard architecutre than the other candidates for a standard architecture. The UYK-41 and UYK-19 still benefit from commercial investment in the support software, and it is only at very high levels of government investment that the architectural efficiency of the UYK-20 brings its life-cycle costs to those of the UYK-19. Figure 5 shows the effects of a lower logistics cost for support software than the nominal value. When logistics costs are lower, a greater fraction of investment dollars can be used to purchase new tools. This perturbation of the nominal values tends to diminish the effects of present size of software base on life-cycle costs, and tends to emphasize architectural efficiency. Here the UYK-20 catches up to the UYK-19 at a much lower level of investment. Again the UYK-41 and UYK-19 ar favored at lower levels of government investment because of the assumed commercial expenditures for the maintenance of part of the software base.

V. Summary and conclusions

The model and data presented here provide for a very strong case for standardization of some kind in military computer systems. The model indicates that maintaining multiple architecture results in billions of dollars of extra expenditures over the twenty-two year life cycle studied here. Whatever gains might be attributed to local optimization by selecting the "best" architecture for each system are lost on a global level when one considers the costs required to maintain and develop sufficient tools for each architecture to have high programmer productivity for applications software.

In looking into the question of which standard architecture to select given that there is a single standard architecture, the model suggests that the UYK-41 will result in the least life-cycle cost, and there will be a substantial savings in this choice over some of the possible choices. In terms of software tools, the UYK-7 and the UYK-41 have roughly comparable bases as measured by the idealized set required for these applications. However, the UYK-7 showed up poorly in the study of architectural efficiency so that it is easteful of hardware to some degree as compared to the UYK-41. The UYK-20 comes out very well on the hardware efficiency measures and is competitive in this respect with the UYK-41, but it lacks both the software tool base and the commercial investment to support part of the tool base. The UYK-41 enjoys both of these advantages. The UYK-19 has a lesser degree of commercial investment than the UYK-41, and has a somewhat smaller tool base, but it generally shows up very well because it has relatively good marks for both tools available and architectural efficiency.

The model's results unequivocally point to the UYK-41 as the architecture that leads to the least life-cycle cost. In second positon it generally favors the UYK-19, but here the results are somewhat sensitive to assumptions, and perturbations could result in the UYK-20 being the second least costly standard. Both the GYK-12 and UYK-7 computers are less desirable as standard architectures than the other choices.

The assumptions of the model are delineated here and described in somewhat greater detail in the full report.[6] The assumptions can be tested and the source data can be scrutinized for accuracy to determine if the model is incorrect or misleading. Presently, a number of aspects of the model are being subjected to further study and refinement. These include:

1. architectural efficiency tests are being repeated with new data points, and should decide whether or not the efficiency measurements are repeatable, and

2. the effects of support software tools on programmer productivity are being reevaluated through the collection of data from a broad range of sources.

We anticipate that some details of the model will indeed change in time. The major conclusion that the UYK-41 leads to the least life-cycle cost is unlikely to change unless new major factors are identified that raise UYK-41 costs significantly more than the other architectures. No such factors are in evidence at present.

References

1. ITEK Corporation, "Preliminary MCF System Configuration Manual," dated 15 June 1977, under US Army Electronics Command Contract DAAB07-C-76-0392.

2. Dietz, William et al., companion paper, this issue of computer.

3. Stone, H. and J. Wagner, "A comparison of existing software bases of the AN/GYK-21 and current military architectures," ECOM Preliminary Technical Memorandum, May 1977.

4. Cornyn, J. J., W. R. Smith, et al., "Life-cycle cost models for comparing computer family architectures," AFIPS Conference Proceedings, Vol. 46, AFIPS Press, Montvale, NY, pp. 185-199, 1977.

5. Svirsky, W., T. Giles, and A. Irwin, "Life-cycle cost analysis of computer family architecture (CFA) finalists within Army emb~dded computer systems," System Development Corporation, unpublished manuscript generated for CFA Selection Committee, August 1976.

6. Stone, H., "Life-cycle cost analysis of instruction-set architecture standardization for military computer-based systems," under contract to U.S. Army Research Office, Contract DAAG29-76-D-0100, January 1978.

7. Wagner, J. et al., "Evaluation of the software bases of the candidate architectures for the military computer family," AFIPS Conference Proceedings, Vol. 46, AFIPS Press, Montvale, NY, pp. 175-183, 1977.

A Proposed System of Buses for the Military Computer Family

William E. Burr

Communications Research & Development Command
U.S. Army

The Army has over one hundred tactical computer based systems under development; however, a few of them have yet been fielded. The cost of developing these systems is proving to be high, and the introduction into the Field Army of many different types of military computers, each with its unique maintenance procedures and spare parts requirments, promises to place a heavy burden upon the Field Army. In addition, the Army has been roundly criticized for failing to achieve competition in military computer procurements, and for buying and deploying computers which are obsolete. The Military Computer Family (MCF) program proposes to standardize on a small set of instruction-set architectures for Army systems, and to develop a strong set of support software (compilers, editors and the like) for these systems as a partial solution to increasing software costs, and to standardize on a set of "modules" from which a wide range of computer systems may be assembled, as a solution to the logistical and maintenance problems of the Field Army. Moreover, these modules are to be specified and procured on a form, fit, and function basis, permiting full competition throughout the MCF life-cycle, and the graceful insertion of improved technology, by the replacement of obsolete modules.

The standardization upon modules, in turn, requires that standard interfaces by developed to connect the modules. These interfaces take the form of a system of standard computer buses. This paper described the general interconnection or bus strategy proposed for the MCF, which is similar in many respects to that used by several successful manufacturers of "mini" or "midicomputers", that is a single, unified system or backplane bus, which ties together various modules and the peripheral buses which link peripheral devices to the system bus. This approach is modified slightly, by the inclusion of a special interchassis bus which ties together the system buses of two separate chassis. This is done because of the necessity to package military computers in small chassis, and to operate in very noisy environments.

A PROPOSED SYSTEM OF BUSES FOR THE MILITARY COMPUTER FAMILY

William E. Burr
US Army Communications Research and Development Command
Ft. Monmouth, N. J. 07703

15 September 1978

INTRODUCTION.

The Army has over one hundred tactical computer based systems under development, however few of them have yet been fielded. The cost of developing these systems is proving to be high, and the introduction into the Field Army of many different types of military computers, each with its unique maintenance proceedures and spare parts requirements, promises to place a heavy burden upon the Field Army. In addition, the Army has been roundly criticized for failing to acchieve competition in military computer procurements, and for buying and deploying computers which are obselete. The Military Computer Family (MCF) program proposes to standardize on a small set of instruction-set architectures for Army systems, and to develop a strong set of support software (compilers, editors and the like) for these systems as a partial solution to increasing software costs, and to standardize on a set of "modules" from which a wide range of computer systems may be assembled, as a solution to the logistical and maintenance problems of the Field Army. Moreover, these modules are to be specified and procured on a form, fit, and function basis, permiting full competition throughout the MCF life-cycle, and the graceful insertion of improved technology, by the replacement of obselete modules.

The standardization upon modules, in turn, requires that standard interfaces be developed to connect the modules. These interfaces take the form of a system of standard computer buses. This paper describes the general interconnection or bus strategy proposed for the MCF, which is similar in many respects to that used by several successful manufacturers of "mini" or "midicomputers," that is a single, unified system or backplane bus, which ties together various modules and the peripheral buses which link peripheral devices to the system bus. This approach is modified slightly, by the inclusion of a special interchassis bus which ties together the system buses of two separate chassis. This is done because of the necessity to package military computers in small chassis, and to operate in very noisy environments.

THE MILITARY COMPUTER FAMILY.

The first premise of the Military Computer Family was that the Army should standardize upon a single instruction-set architecture, or as small a set of instruction-set architectures as possible, for tactical computer applications and then implement a family of militarized computers which execute that instruction-set with various levels of performance. Expected benifits included:

.Software Transfer. Wide transferability of fielded software (eg real time operating systems, utility routines, I/O interfaces, communication control software, etc.), including machine oriented or assembly level code.

.Support Software Base. The ability to create a single, extensive support software base (eg. compilers, editors, debuggers, etc.) to support cost effective software development.

A joint Army/Navy Architecture Selection Committee considered a number of alternative instruction-set architectures, and, in August of 1976, chose the Digital Equipment Corp's PDP-11* instruction-set architecture, which for the purposes of the MCF has been given the military nomenclature AN/UYK-41(v), as the best choice for the Military Computer Family (BURR77).

A second major consideration in the MCF was the need to support a few existing military instruction-set architectures in the Military Computer Family, probably via some form of emulation. The reasons for this are very similar to IBM's reasons for supporting emulations of their second generation machines on their S/360 family; the Army has invested large sums in the development and testing of applications software, much of it in assembly language, for these instruction-set architectures, and, even though few of these systems have yet been fielded, is understandably reluctant to abandon this investment.

A third major consideration in the Military Computer Family program is that intensive competition is desired between multiple suppliers throughout the MCF life-cycle. Past military computer developments have generally been competitive only in the development phase; production contracts are nearly always awarded to the winner of the development effort.

In theory, competition can be acchieved by a "build to print" approach. The original designer is paid to produce engineering drawings, masks, and other appropriate documentation, and then other companies are asked to manufacture copies of the original design. This approach minimizes the spare parts problem, since every card and component is identical, whomever the manufacturer may be, but it has rarely been successful with military computers, because building to print involves close cooperation between compeeting suppliers, and often requires the sharing of proprietary manufacturing processes. The strongest disadvantage to this approach, however, is that it completely stifles technical innovation and is in direct conflict to another major MCF goal, the ability to gracefully inset new technology in existing MCF computer systems.

The fourth major consideration has been technology insertion. The long development cycle and life cycle of typical military computer based systems, in combination with the very rapid pace of component technology development, has meant that military computer hardware frequently is obselete before it is fielded, and systems which have been the field a few years often are almost museum pieces. The repair parts for these systems often become virtually unobtainable at any cost. The Navy, for example, has a large number of computers, in critical fleet applications, which still use germanium transistors. The MCF appraoch provides for the simple graceful and piecemeal insertion of new technology in existing systems by breaking the computer systems down into a set of standard modules, connected by a system of standard buses; as new technology permits each module can be independently replaced by a new improved module.

The fifth major MCF goal is to develop a maintenance approach which the Field Army can live with. The volunteer Army simply does not attract any large number of very intelligent enlisted men, who can be expected to diagnose and repair a wide range of highly integrated computer hardware by reading complex maintenance manuals and using general purpose test instruments. Also, the need to stock large numbers of different repair parts in forward areas can create an intolerable logistical burden. No computer system is of much value to the Field Army unless it can be maintained by more or less ordinary soldiers, and has an acceptable spare parts burden.

The MCF solution to the dilemma presented by the need to provide for competition throughout the MCF life-cycle, to allow the insertion of new technology throughout the MCF life-cycle, and to simplify the maintenance and logistical burden presented to the Field Army, is to develop form, fit and function specifications for a modular system of elemental, interchangable, MCF parts, which may be procured on a competative basis. These interchangable MCF parts are called "MCF Modules" and they will be connected together by a system of standard MCF buses, which provide a uniform interface for intermodule communication. Precisely what an MCF module is will be further explained in the following section. Manufacturers will be free to use any technology they wish to implement MCF Modules, as long as they meet the module form, fit, and functional requirements. Although there may be several manufacturers for a particular MCF module, they will be interchangable, thus reducing the spare parts burden. Moreover, the modules are carefully partioned to permit the configuration of a wide range of systems from a relatively few module types, to allow self-diagnosis of the MCF computer system to a faulty module, and to permit a reasonable cost for spare modules (typically $1k to $10k).

THE PROPER BLACK BOX.

A form, fit and function specification views the items it specifies as a "black box," whose external functional characteristics, interfaces, dimensions and so on are known, but whose internals are left up to the vendor. Such form, fit and function specifications have been highly successfully applied to commercial avionics equipment (BOER74, SMIT76, and GRAH75). An "MCF Module" then is such a black box. Such specifications may be drawn at different levels, and just what level at which they are drawn is an issue of considerable importance. For example, the standard buses needed for the interconnection of modules are largely determined by the size of the black box and the type of packaging chosen. Figure 1 shows a hierarchy of levels, starting at the top with the Army's planned Integrated Battlefield Network, and proceeding down through seven levels to the individual gates on IC's. Standardization is potentially possible at any level between Integrated Circuits and Single Mission Systems. For the Army the strongest considerations probably have to do with logistics and maintenance. It is fairly obvious that the module form, fit and function specifications should be drawn to the level of the Least Replacable Unit (LRU), otherwise each module would have its own separate maintenance requirements and spare parts float. With this in mind the following considerations apply:

.Number of Module Types. The choice of modules should minimize the number of different types. In general the lower the level, the fewer LRU's which are theoretically required. For example if the module were an Intergrated Circuit, then any posible logical function could be implemented from one Nand or Nor circuit type. Obviously the effect on density would be severe. Morover, the effect on technology would be stifling.

.Technology Limitation. The larger the module, the less the limitation on technology, and the greater the freedom of vendors to exercise their ingenuity to design better units. Also, the.larger the module, the fewer the standard interfaces (buses) required.

.Cost. The larger the module, the more expensive and less reliable it (but not the overall system) will be. Consider, for example, an entire memory chassis with 512K bytes of memory. With present technology such a spare part would cost on the order of $50K. The storage capacity would be excessive for many systems. Several different performance levels might be needed, and there would be requirements for both semiconductor (for speed and maximum density) and magnetic core (for nonvolatile memory) versions. Yet it is easy to partition the chassis into a memory controller or bus interface, a chassis, a power supply, and several types of "memory modules" of 64 to 128k byte capacity. The power supply might well be shared across several different chassis types. The individual memory modules will be considerably less expensive and more reliable than the entire chassis. Moreover, they can also be shared with chassis which include both a CPU and memory, which will also be needed. The failure of a single chip in the chassis will not force replacement of an entire $50K chassis, and it may even be economic to throw away defective modules.

.Number of Interfaces. An important consideration is the number of interfaces required to support the partitioning into modules on a form, fit, and function basis. For reasons of configuration control, simplicity, flexibility, and interoperability it is very desirable to minimize the number of different interfaces for modules. These interfaces can be reduced by the use of as few standard buses as possible for the interconnection of all modules.

The solution chosen by the MCF is a compromise: it is to draw the line at well defined functional entities which can easily plug into a single processor-memory bus. This is very similar to the approach taken by a number of manufacturers of minicomputers and microprocessors, who offer bus oriented product lines. For example, the DEC UNIBUS* is used in the PDP-11 line to accommodate a number of different CPU's and interconnect them with memory, a wide variety peripheral devices, DMA controllers and the like. The INTEL Multibus provides a standard framework for the interconnection of various microprocessors, memory, and various support chips(INTE76). The Honeywell MEGABUS** uses an ingenious "split-cycle" protocol to increase bus bandwidth, to reduce the effects of memory latencies on system performance, to permit very flexible interconnections of system elements, and to accommodate a wide range of processor power on a single "unified" bus structure (CONW77, and CASS76). The S-100 bus, despite a lack of a definite standard (a standard has been proposed, MORR78), and several serious shortcomings, has become a very widely accepted standard for "personal" and small business systems; as of the summer of 1978 more than 30 manufacturers offer complete computer systems using this bus and over 130 manufacturers offer products, such as memory cards, floppy disk controllers, and the like for the S-100 bus (ISAA78 and OGDI78). Other semiconductor manufacturers have also introduced more or less standard component level buses for microprocessors (FORC78).

The MCF modules, then, are form, fit and function black boxes, such as a CPU, a DMA controller, a Bus Extender, or a Memory Module, which can be effectively connected by a standard processor-memory bus and interface. A "module" is not synomous with a "card", rather it is an assembly of one or more cards, which is connected to the standard bus(es) and which is treated in the field as a LRU. The planned MCF Modules are summarized in Table 1. Any lower level of standardization would dramatically constrain technology (imagine, for example, the effect of partitioning CPU's into "standard" ALU, control sequencer, and microstore cards), but higher levels would introduce unacceptably large, expensive, and complex LRU's, and probably require more unique kinds of LRU's. This level of partitioning has become fairly conventional, is well understood, and is amenable to effective interconnection by a conventional processor-memory bus. It easily accommodates increased levels of integration over time, by either increasing the capacity or performance of the modules, by reducing their size, or by combining their functions. Moreover, the cost of equivalent modules will undoubtedly come down in the future, so many of these modules may soon become "throw away" items in the future, which will further reduce the logistical burden in the field. Finally, this relatively "coarse" partitioning of LRU's greatly simplifies fault isolation in the field; it is much easier to determine, for example, that the CPU is not functioning properly, than it is to determine what particular part of the CPU is bad.

There are many prices paid for this sort of standardization. The uniform bus interface and structure itself necessarily limits performance and constrains technology somewhat. The desire to have a minimum set of Least Replacable Units makes those units less than optimal for specific applications. On the other hand, any system which cannot be maintained where it counts, by soldiers on a battlefield, is worse than useless, it is a burden which detracts from the Army's ability to fight.

PACKAGING.

The MCF has settled on ATR type packaging for several reasons:

.Existing Standard. ATR cases, which measure approximately 6" by 9" in cross section and have various lengths from 12" to 20", are already a widely accepted packaging standard for many electronic systems, including commercial and military avionics equipment, and both ground and airborne military computers (ARIN74).

.Size. The ATR case is a convienient size suitable for many avionics applications, and its relatively small size also makes ATR cases a one or two man load for ground applications.

.Card Size. The 6"by 9" cross section makes a convenient card size, which is big enough to hold quite a bit of logic, with military packaging techniques, and small enough to be mechanically tractable. It is possible to package modest performance CPU's on a single ATR size card today, and higher performance CPU's on two to twelve such cards. In the relatively near future it will be possible to put a medium performance computer and some local memory on a single ATR size card.

However, the ATR case does carry with it two serious penalties. First, it is too small, for the next few years or so, for optimal implementation of high performance large memory systems. Therefore, it is necessary to break such systems up into a number of separate chassis, and this partioning inevitably causes performance penalties. Secondly, all external bus connections must be made through a 6" by 9" front panel, which makes realestate for interconnection very limited indeed.

PERFORMANCE RANGE.

The performance range to be supported is, of course, a major factor in the design of a bus system. Current MCF plans call for CPU's in three performance ranges:

.low	150-250 KOPS
.medium	250-400 KOPS
.high	400-1000 KOPS

In addition, we also wish in the future to be able to implement CPU's in the 1 to 3 MIP range. It is recognized that, in order to effectively utilize such high performance CPU's, a larger than ATR size chassis may be required, and it may be necessary to employ either some cache memory, or parallel processor-memory buses.

In short, the MCF plans in the near term to implement CPU's with a performance range of about 1:6, and in the long term of 1:10 or more. This is an ambitious range; using the UNIBUS Digital Equipment Corp. implements, at the low end, the PDP-11/04, and at the high end the PDP-11/60. The performance of the 11/04 is on the order of 250 KOPs, and that of the 11/60 is on the order of 630 KOPs, a range of roughly 1:2.5 (SNOW78). To acchieve this performance the 11/60 resorts to a cache memory system (SNOW78 and MUDG77).

The use of CPU's with cache memories poses several concerns, in addition to the stale data problem. The loads presented to the bus system by two CPU's of nominally equivalent performance, one with, and one without, a cache memory, may be very different. Cache performance deteriorates when there are very frequent context changes. Cache memories cannot, in general, be relied upon to enhance performance when responding to interrupts with very time critical latencies, because the cache memory probably will not be well conditioned when the interrupt is recieved. Nevertheless, cache memories and/or multiple processor-memory buses may be required at the high performance end. Although cache memories present a number of problems, their advantages in unloading the bus system and in reducing the performance effects of long access latiencies, are too much to be ignored, particularly in systems, like the MCF, which are partitioned into relatively small boxes.

CONFIGURATIONS.

Just as the MCF must support a wide range of performance, it must also support a wide range of configurations. The simplest configuration is a single chassis computer, with a low or medium performance CPU, 64 to 256K bytes of main memory, and one or two I/O buses, all housed in a short ATR chassis. Early requirements have also been identified for a uniprocesor system with a medium to high performance CPU and 64 to 256 Kbytes of main memory connected to one or more memory expansion chassis (each containing up to 512 Kbytes of memory) and to an I/O expansion chassis. Figure 2 illustrates this configuration. The most complex configuration, for which an immediate need has been identified, is a dual processor configuration shown in Figure 3. In this system there are two computer chassis connected to two private memory expansion chassis, a private I/O expansion chassis, and a shared memory expansion chassis. In normal operation one CPU functions as a message processor, while the other functions as a data base manager. If either system fails, then the other performs both functions at a reduced rate.

This is not the full intended range of configurations. Although specific programs have not yet been identified for such configurations, multiprocessor arrays of various kinds will undoubtedly be desired for future Army systems. An example would be a system where a number of general purpose CPU's, possibly with integral cache memories, may be housed in the same chassis, and connected to a common main memory. The design of general purpose operating systems for such systems remains something of a research issue, but if it can be solved satisfactorily, the abliity to enhance performance by adding CPU's, and the ability to survive CPU failures would both be attractive properties. Most contemporary computer bus architectures are organized around a single relatively high performance processor, but the dramatic reduction in the cost of processors will surely lead to other organizations.

In fact, the development of powerful but inexpensive microprocessor chips offers the attractive possiblity of connecting large arrays of small inexpensive processors together to form very powerful configurations. As an example, a typical military command and control system can be broken into some sort of communications control function, an overall system management function, one or more applications functions, and a data base management function. Figure 4 illustrates how a set of dedicated microprocessor based modules, each with an integral program ROM, and scratchpad RAM, might be coupled together to create such a system. The idea of modules with a dedicated function appears to be in conflict with the desire to minimise the number of unique module types, however, many such modules could serve many different battlefield systems. Potentially, such a system could be highly redundant, and therefore resistant to module failures, and be expandable in a modular fashion. It is uncertain which kind of interconnection architecture is most advantageous for such systems, or how tightly processors should be coupled in such systems, but a bus standard for future military systems should make some explicit provision for the interconnection of relatively large arrays of processors.

BUS TAXONOMY.

A functional taxonomy for MCF buses is given in Table 2, and Figure 5 shows a simple multichassis system using these buses to connect some of the modules listed in Table 1, peripheral devices, and chassis. Four kinds of buses are defined:

.Internal Bus (I-bus). The I-bus is essentially a backplane bus which is the standard interface "seen" by all MCF modules. The I-bus never extends outside a chassis.

External Bus (E-bus). The E-bus joins the I-buses of different chassis. The major difference between the E-bus and the I-bus is the type of circuits employed; the E-bus requires differential circuits to reduce common mode noise problems.

.High Speed I/O Bus. This is a high bandwidth block transfer oriented I/O bus.

.Low Speed I/O Bus. This is a moderate bandwidth, word at a time oriented I/O bus, intended for relatively low speed word or character oriented devices.

Many other taxonomies are possible. It is certainly possible to combine the functions of several of the buses. For example, the UNIBUS functions as both a processor-memory and a low speed I/O bus in commercial PDP-11's. The E-bus and I-bus might also be combined at the price of forcing every module to use relatively high powered differential drivers (overall bus length would likely limit performance if this were done). Similarly, the processor-memory bus might be split in a number of ways; separate buses might be used for operand and data acesses, for I/O control functions, and for interprocessor synchronization and signaling (interrupts). This would improve bandwidth at the expense of complicating the interconnection and interfacing of modules, and of increasing the number of required wires and interconnection points. Changing the size of the black box would also change the bus taxonomy. For example, if the black box were an entire chassis, rather than a module, then there would be no need for an I-bus standard. Similarly, if the black box were the entire computer system, then only peripheral bus standards would be required.

BUS EVALUATION.

Rather than devise a new bus structure for the MCF, an attempt was made to find a existing buses to satisfy the MCF needs. This had been the MCF approach to selecting an instruction-set architecture, and this approach had been apparently successful for that purpose. Due to its use in separate Army and Navy major computer development efforts, a selection was made, for planning purposes, of the bus used in the CDC 480 family of military microprogrammable processors. This selection was strongly criticized by industry as being inappropriate and anticompetative (ROBE77). In response to this criticism an MCF Computer Bus Architecture Evaluation Committee was formed, and a number of buses, including the 480 bus, were examined for their suitability as each of the four bus types shown in Table 2. In making this evaluation, it was assumed that military versions of the various candidate buses could be produced; that is the buses would be modified to meet military requirements. For example it was assumed that signals might be converted, where necessary, to differential form, and that address and data lines could be extended in width as necessary.

In the end this effort was not successful because:

.Availability of Data. The data avaliable on the various buses examined varied greatly in its quality, form, and completeness. Often it was necessary to "read between the lines" when attempting to identify bus characteristics. Moreover it was not posible to obtain data on some recent, interesting buses, which are proprietary. There was no assurance that the rights to most of the buses considered could be obtained.

.Modification. The assumption that a bus could be modified to suit military environments is questionable. For example, some buses which use wire-ored open collector signals sometimes allow transitional states where one driver may be attempting to drive a line high while another driver may be driving it low. This causes no problem with open collector circuits, but would with tristate or differential drivers.

Cohesion. Attempting to independently select different buses for each function defined in Table 2 simply did not result in any cohesive bus system.

Multiprocessor Support. Few of the buses examined, including the 480 bus, made any explicit provision for supporting multiprocessor or distributed processor configurations, yet there is great interest in such systems for future military systems. Clearly, a bus system for future military applications should provide an explicit framework for connecting multiple or distributed processors.

As a result, the Bus Selection Committee reccommended that, rather than attempting to use existing buses, a new cohesive bus structure should be developed for the MCF.

MCF BUS SYSTEM REQUIREMENTS.

Since the Bus Selection Committee was unable to select a suitable set of existing buses, the solution chosen has been to develop functional requirements specifications for each of the four types of buses shown in Table 2, and to ask bidders for the initial MCF systems implementation contract to propose specific bus structures for the MCF. Design considerations and requirements for each of these buses are summarized in the following paragraphs.

I-BUS.

The I-bus will be the primary interconnection interface between MCF modules. Since there will be an I-bus interface in every MCF module (except power supplies), it is vital that the interface either be simple to implement in a modest number of SSI, MSI or standard programmable LSI circuits (PLA's, ROM's, RAM's, PMUX's, etc.), or that an appropriate custom LSI interface be developed to support this interface.

Although the I-bus corresponds functionally, in many respects, to a conventional computer backplane, unlike the usual backplane it is a very regular, general purpose communication path, and must be functionally independent of the particular processor, module, or memory mix. Some sort of programming function will probably be required to define bus configurations and module addresses, however special system specific point to point wiring changes on the backplane are not allowed. The I-bus need not support physical lengths greater than 1M; this is sufficient to allow for expansion beyond the ATR case size.

The I-bus may be either sysncronous or asyncronous, however devices of various latencies must be accommodated. The bus should support high bandwidths and short latencies for transactions which take place entirely within a single chassis. To improve bandwidth the I-bus could be broken into two or three independent buses, for example an I/O control bus, an event (interrupt) bus, and a data bus, however this would complicate interconnections, and increase the number of required connector pins and bus signal lines, and probably would not dramatically improve performance. There are at least two "unified" processor memory buses in use today which approach or acchieve the desired level of performance, the Honeywell MEGABUS, used in the Level-6 computer family, and the System Backplane Interconnect (SBI*) bus used in DEC's new PDP-11 32-bit virtual address extension/enhancement, the VAX-11/780 (DEC77). The MEGABUS is a an asysncronous bus which supports a 24-bit address space, and a 10-bit processor address space. It uses a proprietary distributed bus mastership algorithm. The SBI bus is a sysncronous bus with a 200ns cycle time and a length limited to 2M, which has a 32-bit multiplexed address/data path and ordinarily transfers 8-bytes of data on 2 consecutive bus cycles. It acchieves an overall transfer bandwidth of 13.33 million bytes per second (the UNIBUS nominally acchieves 1.5 Mbytes/sec.). Interestingly, both the MEGABUS, which is asynchronous, and the SBI bus, which is sysncronous, employ a "split read" cycle, which requires two separate bus cycles to accomplish a read, and frees the bus during access latencies. This has two advantages and two disadvantages: it increases overall bus bandwidth, and makes bus bandwidth essentially independent of memory latencies; on the other hand it makes for a more complex interface and tends to increase best case access latencies somewhat. Both buses also employ a distributed mastership scheame. Unfortunately both DEC and Honeywell were understandably unwilling to release the details on these very recent and interesting buses to the bus selection committee, so they were not included in the MCF bus evaluation.

A split cycle appears to be desirable, but not essential, if I-bus bandwidth requirements are satisfied. A split cycle does seem particularly advantageous for multiprocessor configurations, since several processors can interleve simultaneous bus transactions. MCF performance requirements probably will force a 32-bit data path, and the desire to be able to accommodate very large future memories indicates that a 32-bit address space should be provided for as well. Although it is not necessary on the I-bus, since connectors with large numbers of pins can be accommodated with ATR size modules, it may be most satisfactory if address and data are multiplexed. This would add a little logic to each interface, but would reduce connector pins. Moreover, it costs bandwidth only on write operations, and the SBI bus demonstrates that high bandwidths are attainable with a multiplexed bus.

A simple linear topology is desired for the I-bus, but it is necessary that it be possible to remove or power down functionally noncritical modules from the bus without affecting bus operation. Either a distributed bus mastership scheame, or a redundant bus arbiter is desirable to insure that the failure of one master arbiter does not bring down the entire bus. An incidental advantage of a distributed mastership scheame is that mastership signals need only propigate in one direction, no reply is required from an arbiter, so mastership latencies can be reduced.

E-BUS.

The E-bus probably represents the most difficult design problem of any of the MCF buses. It runs between chassis and connects their I-buses. Modules in separate chassis must be able to communicate with each other as if they were in the same chassis. Each signal on the I-bus requires only one wire, but noise rejection requirements will probably force the use of two wires per signal and differential circuits on the E-bus. Connector space on the 6" by 9' front pannel is very limited, and shielded military cables are both bulky and very expensive. Moreover the same logistical requirements which limit the number of module types, also argue strongly in favor of just a very few standard cable lengths and types. It is unacceptable for each different Army system to requi.e a special cable set. Signal conversions are required at either end of the bus, causing timing delays, and the bus may be as long as 5 Meters. A linear interconnection approach is desired with two connectors on the front of each chassis and a "T" connection with a simple conduction path between the two connectors. It must be possible to connect at least five chassis to one E-bus. Such a bus is effectively a transmission line and there are impedance matching and reflection problems to overcome. Signals could be regenerated in each chassis, this would permit the interconnection of more chassis, but would introduce additional propagation delays and the failure of any chassis would break the bus. An alternative might be some sort of star connection; this would reduce overall bus length, but require signal regeneration for communication where neither of the chassis is the central node, and would have to find a way around the limited front pannel connector space. Moreover, the central node would be a point of vulnerability; its failure would bring down the whole system.

Limitations on connector and cable size will probably force multiplexing of address and data signals. Moreover, conversion and propigation delays will also limit bandwidth. The E-bus is likely to be the performance bottleneck in MCF systems. The use of a split cycle read protocol might considerably reduce this problem. To allow for effective use of multiprocessors the Bus Interface Modules (BIM's), which connect E-buses and I-buses might buffer transactions and independently arbitrate E-bus and I-bus transactions. This would permit a number of I-bus transactions to go on within separate chassis, in parallel with an interchassis transaction on the E-bus, but would introduce possible deadlock problems. Also, when building uniprocessor systems with several memory chassis, the need to rearbitrate E-buses and I-buses may introduce long latencies. A possible solution might be to introduce two different BEM's, one which buffers transactions and one which does not, for different types of systems. This, of course, is in conflict with the desire to minimize the number of different module types. It would also be possible to implement a split read cycle on the E-bus, where bandwidth is critical, but use a single read cycle on the I-bus, where bandwidth is not so hard to get. It would also be desirable to allow some sort of short block transfer or extended word transfer mode on the E-bus to increase its bandwidth. If this were done, it might also be advantageous to build some sort of buffer or cache memory into the BEMs as well, as an optional feature.

The design of the E-bus is probably the most difficult issue for MCF bidders. The one saving factor may simply be that only BEM's interface to the E-bus, and it is therefore possible to introduce several protocols at the expense of one new module type per protocol, or to provide performance enhancements, like a cache memory integral to the BEM, by the introduction of a different -nodule type, without affecting the interfaces of other modules. This might ultimately allow a graceful transition to some sort of fiber optic E-bus, when that technology matures.

PERIPHERAL BUSES.

One obvious approach to the selection of peripheral buses for the Military Computer Family would be to use the existing peripheral buses of each of the instruction-set architectures to be implemented by the MCF. Current MCF plans call for the development of CPU modules for three different instruction-set architectures: the AN/GYK-12, which is used in several major army systems including TACFIRE, the AN/UYK-19, which is used in a number of planned Army systems and which is an extension of the Data General NOVA instruction-set architecture, and the AN/UYK-41, which is the instruction-set architectjre chosen by the Computer Family Architecture Selection Committee, that is the PDP-11/70. Between them these three machines use four different peripheral buses. Such a system would mean that each MCF peripheral device would need to interface to at least three different buses, and at four different cable sets would probably be required. This is unacceptable, moreover each of these buses is a relatively old design, and only the AN/GYK-12's Peripheral Bus was originally designed for military environments.

Rather than use different peripheral buses for each instruction-set architecture, the decision was made to use two standard peripheral buses for all MCF systems. One peripheral bus, the High-Speed Peripheral Bus is designed for high bandwidth block oriented peripherals, such as disk drives. The other peripheral bus, the Low-Speed Peripheral Bus, is oriented to lower bandwidth "word at a time" peripherals, such as terminals or printers. This approach will eliminate the need for three separate device interfaces for each peripheral device, and will reduce the number of required cable types, but it will mean that earlier peripheral devices will not interface to MCF systems. Moreover, it may still be necessary to use three different architecture specific peripheral Bus Interface Modules (BIM's) to account for differences in the I/O structures and semantics of the three instruction-set architectures. Finally, it is not likely that the same peripheral device can simultaneously interface to the I/O protocols of three different instruction sets, and still preserve the same software interface as some earlier peripheral device. This means that low level device driver software for existing systems will have to be modified if those systems are transported to MCF hardware.

The basic requirements for the two peripheral buses are fairly similar. Minimizing the number of signal lines for each bus is vital, and there is probably a strong case to be made for serial buses using either coaxial cable or twisted pairs, as required, or even fiber optic cables. Both buses should operate over relatively long distances (at least 30 M and more would be desirable). The low-speed bus should be able to concurrently multiplex transfers from a minimum of 31 devices at a rate of 19.2 baud each, while the high-speed bus should be able to address at least seven peripheral devices, with an unmultiplexed block transfer rate of 10 Mbytes per second or more. If any peripheral device on either bus is turned off or removed from the bus it should not affect communications between other devices. It is desirable that bus priorities for both buses be independent of position on the bus. It may be possible, and if so it would certainly be desirable, to combine the two peripheral buses into a single bus type. Even if performance (that is maximum bandwidth) suffers somewhat when this is done, systems with modest I/O requirements could use a single bus while systems with heavy I/O requirements could use two or more peripheral buses.

CONCLUSION.

Any proposed standardization program is likely to be controversial, and the MCF is no exception. In the case of the MCF, the desire to have intensive competition and technology insertion throughout the life-cycle of MCF systems have forced an approach to standardization in terms of modules whose form, fit, and function are specified, but whose internal design is not. The size of modules is a function of the desired packaging, and the desire to have a simple, effecitve maintenance approach, which can be implemented on a battlefield with ordinary soldiers, and with a small spare parts inventory. A set of standard buses is needed to provide standard interfaces between the modules. There are no "free lunches" and various prices are paid for each of these decisions. Nevertheless, however, except for the use of a special interchassis bus, the E-bus, which is necessitated by military noise and packaging requirements, the approach chosen to busing is really quite conventional. Such a "grand bus" approach, where a standard interconnect bus (in the case of the MCF the I-bus and its extension, the E-bus) ties the computer together, is now quite a conventional technique in the mini and midi-computer industry, and the DEC PDP-11 and VAX-11/780, the Data General NOVA, the Honeywell Level 6 computer families all provide a wide product line organized around a common interconnect bus.

The conceptual framework for the MCF bus system has four buses: an internal chassis interconnection bus (the I-bus), a bus which ties the I-buses of separate chassis together (the E-bus), and two Peripheral buses, one for high speed block transfers and one for lower speed word oriented peripherals. The eventual winner of the MCF System Producability Integration Contract will have considerable latitude in the detailed design of these buses, and may choose to combine some of them. With a little luck and a lot of good engineering, MCF bus standards can be acchieved, so that the Field Army can obtain the same advantages of modular interchangability, interoperability, competition, and a large choice of system components that the RS-232 standard brings to data terminals, the IEE-488 standard instrument bus brings to intrumentation, the S-100 bus brings to small business and hobby computers, and buses like the UNIBUS already bring to many users of commercial minicomputers.

* VAX, SBI, PDP, and UNIBUS are all registered trademarks of Digital Equipment Corp.
**MEGABUS is a registered trademark of Honeywell Information Systems, Inc.

TABLE 1. MCF MODULES.

MCF MODULE TYPE	DESCRIPTION
CPU MODULES	Several CPU modules are planned, implementing 3 instruction-set architectures (AN/GYK-12, AN/UYK-19, and AN/UYK-41(V)) over a performance range from 200 to 1000 KOPS.
POWER SUPPLY MODULES	Two general purpose power converter modules are planned.
RAM MODULES	Both semiconductor and core random access memory modules are planned in 32 by 32K or 64K bit sizes.
BUS EXTENDER MODULE	This module is required to convert the basic system backplane bus (I-bus) to a form suitable for communication between chassis (E-bus). Signals will probably be converted from TTL levels on the I-bus to balanced differential signals on the E-bus.
BUS INTERFACE MODULES	These modules connect peripheral buses to the main system bus (I-bus). There may be a different BIM for each instruction-set architecture to accommodate the differing I/O functionality of each instruction-set.
SPECIAL PURPOSE I/O MODULES	These modules provide special purpose interfaces between peripherals and the main system bus. An example would be a Key Generator interface.
SPECIAL PROCESSOR MODULES	No such modules have yet been defined, however obvious possibilities include signal processor modules, data base manager modules, communication line controller modules, and the like.

TABLE 2. MCF BUS SYSTEM TAXONOMY.

NAME	FUNCTION	CHARACTERISTICS
Internal Bus (I-bus)	Standard interface for all modules. Must include processor-memory data, control, and event (interrupt), signal paths. this bus roughly corresponds to a conventional backplane	.Probably TTL signals. .Physically short (< 1M). .High bandwidth. .Short latency.
External Bus (E-bus)	Extension of the I-bus, ties together the I-buses of two or more chassis.	.Differential signals. .Moderate length (< 5M) .High bandwidth. .Moderate latency. .Strongly pin limited. .Expensive shielded cables.
Low Speed I/O Bus	A multiplexed, word at a time bus for low bandwidth peripherals.	.Differential signals. .Moderate bandwidth. .Possibly a serial bus. .Longer lengths (>30M). .Strongly pin limited.
High Speed I/O Bus	A high bandwidth block transfer oriented bus for high speed peripherals such as disk drives.	.Differential signals. .High bandwidth. .Block transfer. .Long latencies acceptable. .Moderate length (> 30M). .Strongly pin limited.

1. NETWORK

2. SINGLE MISSION SYSTEMS

3. SUBSYSTEMS

4. MODULES

5. CIRCUIT BOARD ASSEMBLIES

6. CHIPS (INTEGRATED CIRCUIT ASSEMBLIES)

7. CELLS (GATES)

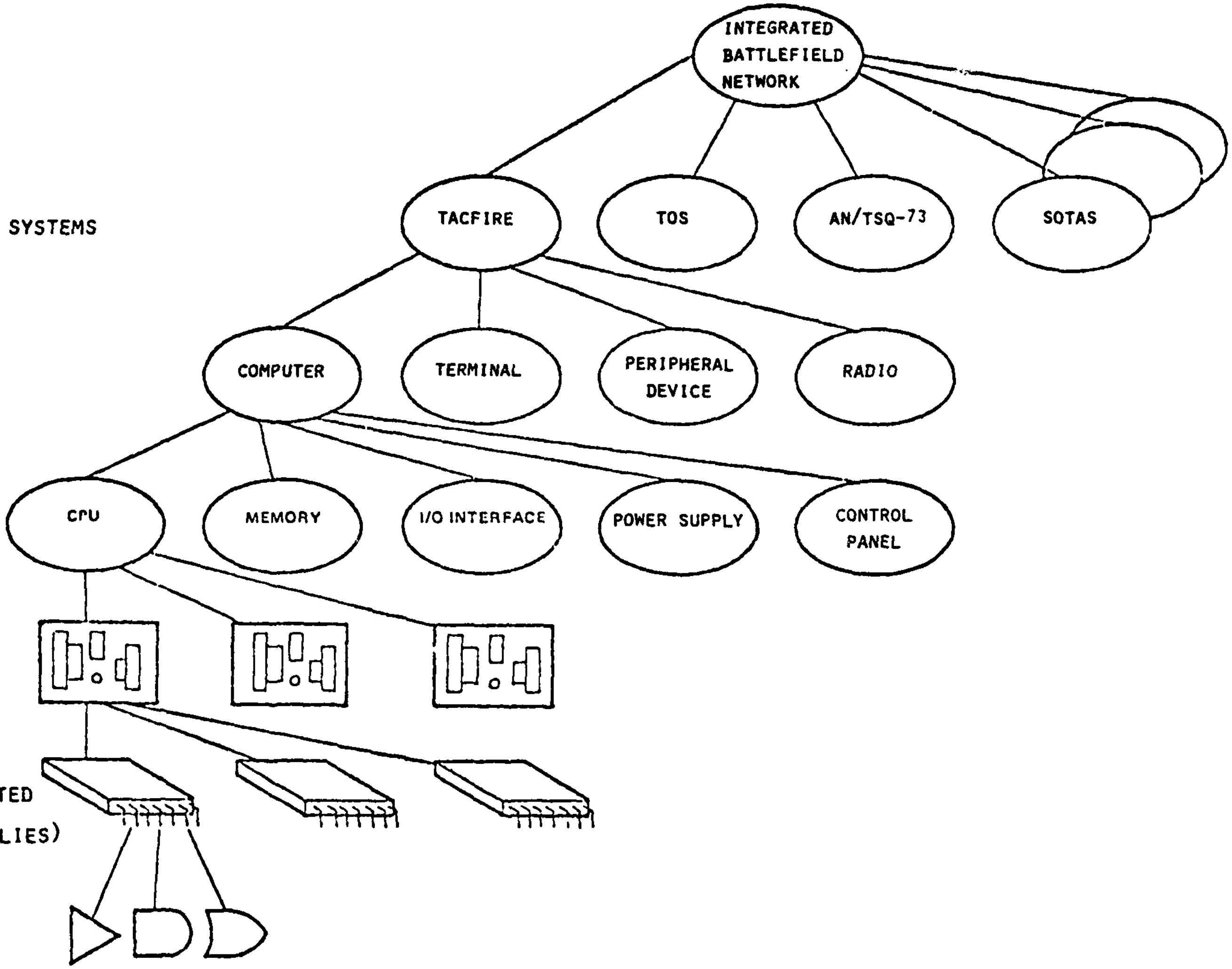


FIGURE 1. AUTOMATED BATTLEFIELD SYSTEMS, HIERARCHIAL LEVELS

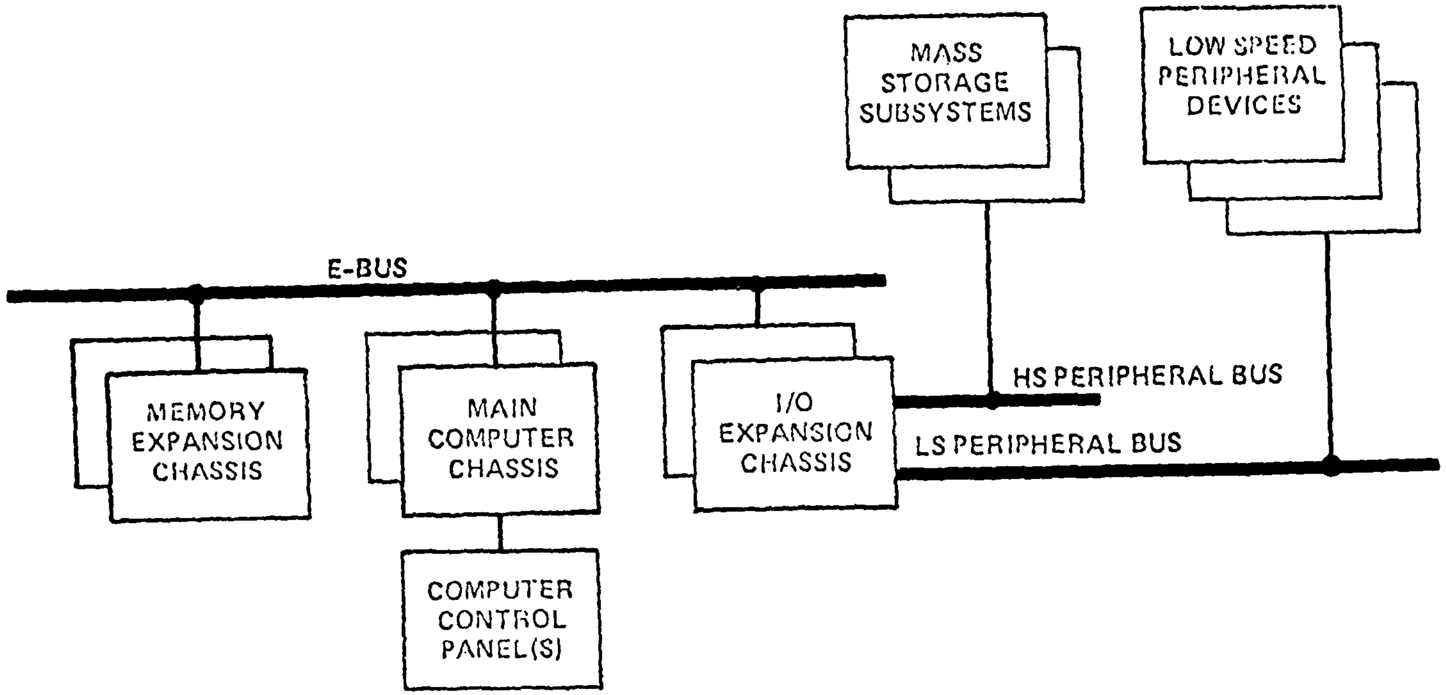


FIGURE 2. MULTICHASSIS UNIPROCESSOR SYSTEM

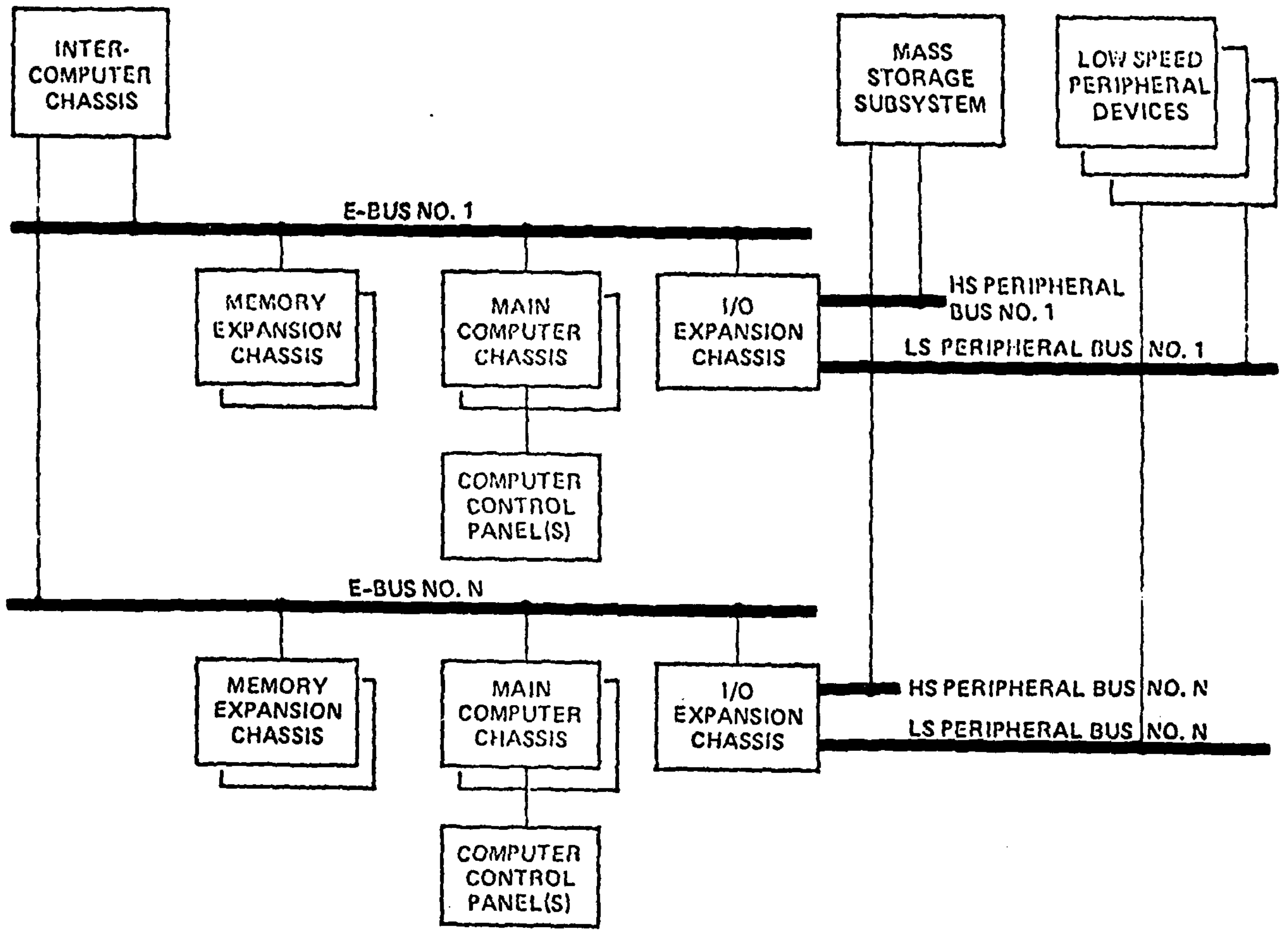


FIGURE 3. MULTICOMPUTER SYSTEM

CENTACS

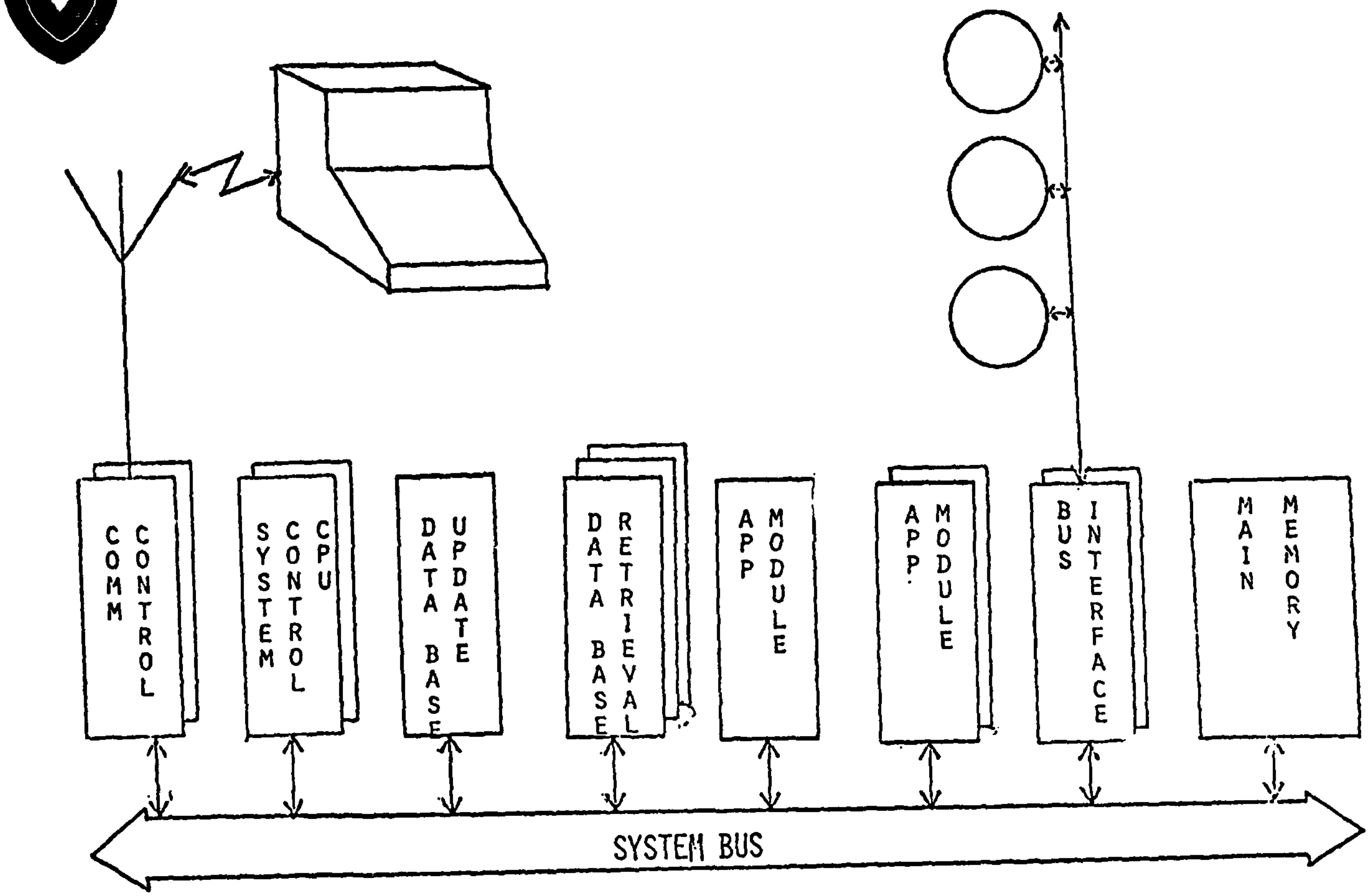


FIGURE 4. POSSIBLE FUTURE C^3 SYSTEM WITH DEDICATED MULTIPROCESSORS.

CENTACS

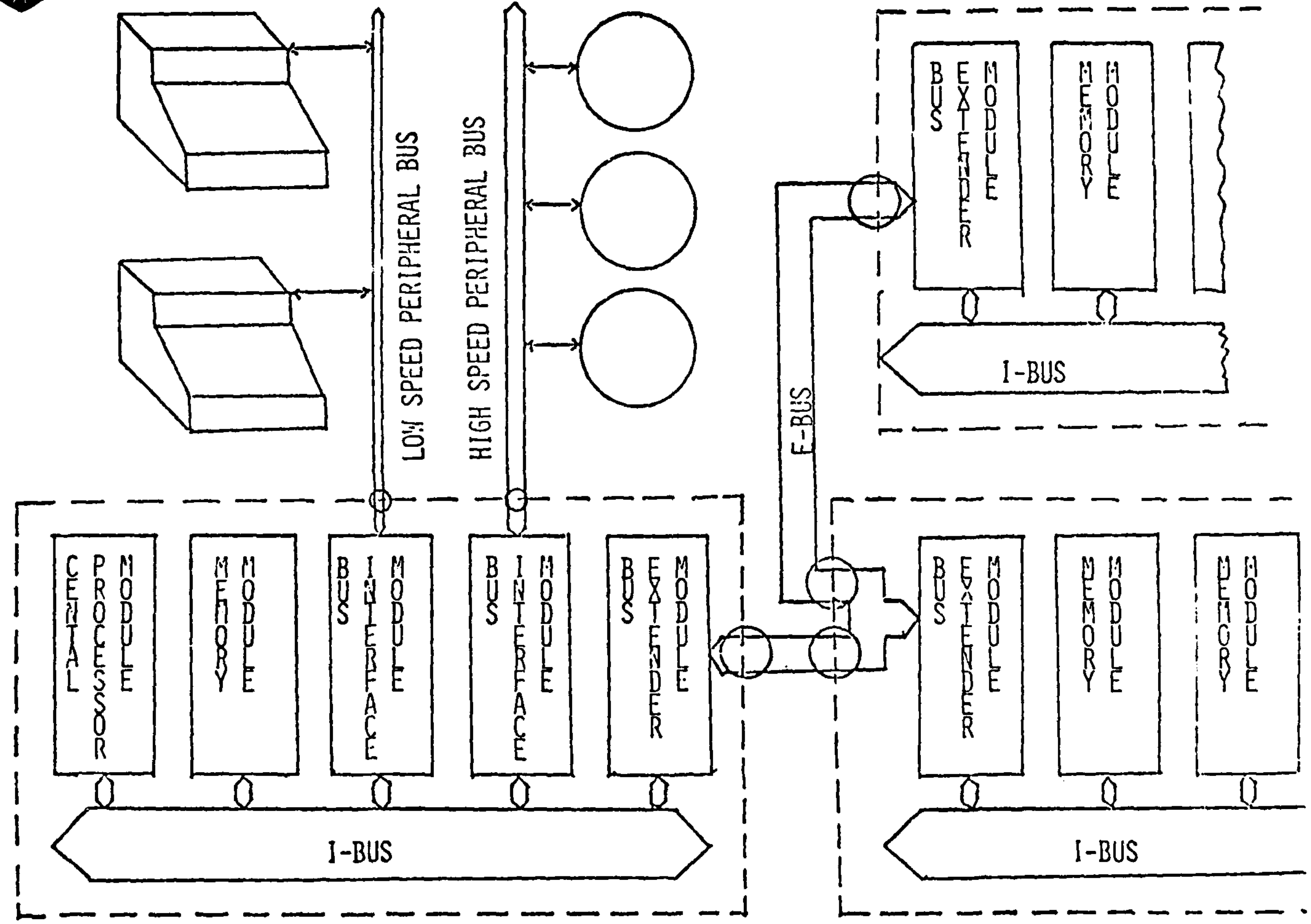


FIGURE 5. MCF BUSES, CHASSIS AND MODULES

REFERENCES

(AERO74) AERONAUTICAL RADIO, INC. "Air Transport Equipment Case and Racking," ARINC Specification No. 404A, Anapolis, Md., Mar. 15, 1974.

(BORI74) Boring, G. and B. Retterer, "Form, Fit, and Function Specifications," TECHNICAL PERSPECTIVE, No. 16, 1974, ARINC Research Corp., Annapolis,Md.

(BURR77) Burr, W., A. Coleman, and W. Smith, "Overview of the Military Computer Family Selection," AFIPS Conference Proceedings, Vol. 46, 1977 NCC, pp. 131-137.

(CASS76) Cassarino, Jr. et. al., "Data Processing System Providing Split Bus Cycle Operation," UNITED STATES PATENT NO. 3,997,896, Dec. 14, 1976.

(CONW77) Conway, J. "Approach to Unified Bus Architecture Sidesteps Inherent Drawbacks," COMPUTER DESIGN, Jan. 1977.

(FORC78) Force, G., "Microprocessor Bus Standard Could Cure Designers Woes," ELECTRONICS, Vol. 51, No. 15, July 20, 1978, p. 113.

(GRAH75) Grahm, L. J., "Application of the Commercial Airline Acquisition Methodology to Department of the Navy Electronic Equipment Acquisitions," ARINC Research Corp. Report, 15 Oct. 1975, Publication No. 1313-01-1-1447.

(INTE76) INTEL Corp., "Intel Multibus Interfacing'", Applications Note, 1976.

(ISAA78) Isaacon, Portia, "Personal Computers for Small Business Applications," POPULAR ELECTRONICS, Vol 14, No. 2, Aug 1978, p. 53.

(MORR78) Morrow, G and H. Fullmer, "Proposed Standard for the S-100 Bus," COMPUTER, Vol. 11, No. 5, May 1978, pp. 84-90.

(MUDG77) Mudge, J. C., "Design Decisions Achieve Price/Performance Balance in Midrange Minicomputers," COMPUTER DESIGN, vol. 16, no. 8, pp. 87-95, Aug. 1977.

(ODGI78) Ogdin, Carol, "Microcomputer Buses," MINI MICRO SYSTEMS, Vol 11, No. 5, June 1978, p97-104.

(ROBE77) Robertson, J., "Government Closeup," ELECTRONICS NEWS, Monday Aug. 1, 1977, p. 12.

(SMIT76) Smith, Noel, "A Marketplace Approach to Military Avionics Standardization," 1976 National Aerospace Conference, Dayton, Oh., May 1976, ARINC Research Corp Publication 6405-1487.

(SNOW78) Snow, E. A. and D. P. Siewiorek, "Impact of Implementation Design Tradeoffs on Performance: the PDP-11, A Case Study," Carnegie-Mellon University Report, CMU-CS-78-104, Feb. 19, 1978.

MILITARY COMPUTER FAMILY (MCF)

MCF BUS SYSTEM CONSIDERATIONS

- LIFE-CYCLE COMPETITION
- TECHNOLOGY INSERTION
- LOGISTICS AND MAINTENANCE
- ATR SIZE CASES
- MULTIPLE INSTRUCTION-SET ARCHITECTURES

CENTACS

MILITARY COMPUTER FAMILY (MCF)

MCF MODULE TYPES

- CENTRAL PROCESSOR MODULES
- RANDOM ACCESS MEMORY (RAM) MODULES
- BUS EXTENDER MODULE (BEM)
- BUS INTERFACE MODULE (BIM)
- SPECIAL PURPOSE I/O MODULES
- POWER CONVERTER MODULE (PCM)

CENTACS

MILITARY COMPUTER FAMILY (MCF)

MCF BUS TAXONOMY

● INTERNAL BUS (I-BUS)	HIGH BANDWIDTH BACKPLANE BUS, RUNS ONLY WITHIN A CHASSIS.
● EXTERNAL BUS (E-BUS)	JOINS I-BUSES OF SEPARATE CHASSIS, RUNS ONLY BETWEEN CHASSIS.
● HIGH SPEED PERIPHERAL BUS	HIGH BANDWIDTH BLOCK TRANSFER I/O DEVICE BUS.
● LOW SPEED PERIPHERAL BUS	LOW BANDWIDTH WORD ORIENTED I/O DEVICE BUS.

CENTACS

Evaluation of Alternative Computer Architectures
An Overview of Three Studies

William Dietz

Department of Computer Science
Carnegie Mellon University

This paper summarized the results of three evaluations of computer architectures conducted by Carneigie Mellon University. As part of the Military Computer Family (MCF) program, methods of measurement were given for program size, memory activity, and processor activity. A description of the statistical design and analysis methods for the experiments was given. Results were summarized for the three evaluations.

EVALUATION OF ALTERNATIVE COMPUTER ARCHITECTURES: AN OVERVIEW OF THREE STUDIES

William B. Dietz

1. Introduction

Until recently, decisions concerning the choice of alternative commercial computer systems were rather straightforward. The list of possible candidates was narrowed down based on questions about the suitability of the system:

- Does it have a compiler for the high order language we are using?

- Does the system have the capacity for the primary and secondary memory required?

- Do necessary tools such as editors and file systems exist?

- Does it have a usable operating system which will support the application?

- Does the system meet any size constraints which may apply?

After the list of candidates has been pruned based on questions such as these, the final selection is usually made based on the time which the system requires to perform either some portion of the application program or a set of "standard" benchmark programs (provided the cost difference between candidates is not too great).

In the last few years a new wrinkle has been added to the scenario described above. We have entered the era of the Computer Family. Now when the salesman from XYZ Computer Systems comes to call, he may not describe just one system he sells which will meet your needs. Instead he may extol the virtues of the XYZ 1000 computer system and how it will meet your current needs, and follow that with a discussion of the XYZ 1000 +1 processor - "Your credibility gap filler". The XYZ 1000 +1 processor can be substituted for the XYZ 1000 processor if you find you need more processing "power" in the future. This substitution can be made, he will assure you, without any changes to time-independent software developed on the XYZ 1000 processor. This creates a dilemma. Which processor do you measure, the XYZ 1000 or the 1000 +1 ? And then the salesman throws in the clincher. He explains that the XYZ 1000 +2 processor is scheduled to be available soon. It will run faster than the XYZ 1000 +1 and will cost less. At this point, the procedure for choosing a computer system must be re-evaluated. The idea and value of a computer family must be considered in the decision-making process.

A computer family is based on relatively simple, yet very important abstract concept, the COMPUTER ARCHITECTURE. The architecture of a computer is defined here to be the structure of the computer that a programmer needs to know in order to write any time-independent machine-language program that will run correctly on the computer. Things like instruction operation descriptions, addressing modes, and data types form a part of the architecture of a computer. However, things like cache memories, pipelining, or parallel data paths are NOT part of the architecture. These ideas belong to the implementation of the architecture.

It is this separation of architecture from implementation that allows a family of computers with varying cost/performance ratios to be built.

The reasons for the interest in computer families are very clear. As the demand for more sophisticated "state of the art" software systems and tools increases, it becomes more difficult (in terms of time and money) to supply this software with new computers designed to take advantage of the rapid advances in hardware technology. If a manufacturer uses an existing architecture for a new hardware implementation most of the existing software can be captured. Also, all of the programmers who are trained and proficient in the use of the existing architecture will be able to generate code for the new hardware without any retraining. Many commercial manufacturers have seen these advantages and have been heading toward compatible computer families for the past several years.

The same forces which cause many original equipment manufacturers to begin using computer families in their systems also affect military systems builders. However, because the suppliers of military computer hardware (and the management of military systems production) are often driven by different forces than the commercial market, a different route had to be taken to try to establish a family of military computers suitable for use in a wide range of applications. The ARMY/NAVY Computer Family Architecture (CFA) committee was formed in 1975 to try and select a single computer architecture which could be used in a wide range of tactical systems. By developing a set of absolute and quantitative criteria, the CFA committee was able to narrow an initial group of nine architectures down to three final candidates. The desire to compare these three candidate architectures lead to the development of the computer architecture evaluation methodology described below. This methodology has also been refined and applied to two other groups of architectures.

The evaluation yields a ranking of the overall relative efficiencies of the architectures as well as some information about their performance in certain subareas. This information along with other factors such as support software availability and applicability was used by Harold Stone to derive a life cycle cost ranking for the architectures. When an architecture is to be chosen for a family of computers that will be used in a wide range of applications, the life cycle cost ranking should be weighed heavily. The architectures being considered must also be examined carefully for deficiencies that could make even a high ranking architecture a liability in specific applications.

2. Architecture comparison method

Three computer architecture comparison studies have been performed at Carnegie -Mellon University. The basic approach which evolved is this. A set of 16 "test program specifications" were developed. Programs from this set were assigned to a group of test programmers tc be coded in assembly language for the architectures being compared. The assignments were made based on a statistical design which permits

FIRST STUDY	IBM S/370
	INTERDATA 8/32
	DEC PDP 11
SECOND STUDY	AN/UYK-7
	AN/GYK-12
	AN/UYK-19
	AN/UYK-20
	AN/UYK-41 (PDP-11)
THIRD STUDY	AN/AYK-14
	AN/AYK-15A
	AN/UYK-41 (PDP-11)
	DATA GENERAL ECLIPSE C/330

Figure 1: Architectures Evaluated

separation of programmer effects from architecture effects in the analysis. After the test programs were coded and debugged using a specified test data set, they were evaluated using three measures of architecture performance: S, M and R.

2.1. The Test Programs

A representative set of benchmark or "test" programs was selected for the evaluation. Small test programs of about 100-500 machine instructions were used due to limitations of budget and manpower. Test programs written in a high-order language were ruled out due to our inability to separate compiler effects from architecture effects. Instead, the algorithms for the test programs were specified as structured programs in a high-order PL/I-like program definition language. The test programmers were instructed to "hand translate" these algorithms into the assembly language of the respective architectures. The programmers were not permitted to make algorithmic improvements or modifications, however, they were free to optimize their code to the extent possible with highly optimizing compilers. Using this method, we hoped to reduce the variations due to programmer skill.

The test program algorithms chosen were intended to be broadly representative of the basic types of operations performed by military computer systems. For the first study 12 test programs were specified. The refined analysis used in the second and third studies called for 16 test programs. The sixteen test programs were divided into four categories for subgroup analysis. These four groups are:

- Interrupts and Traps

- Address Manipulation

- Character and Bit Manipulation

- Miscellaneous

2.2. Assignment of Test Programs

The assignment of test programs was done in accordance with the statistical design of the studies. Each test programmer coded two programs on each of the architectures in the study. The suggested order of writing was different for each test programmer to avoid algorithm/machine familiarity.

2.3. Debugging and Execution Testing

An ISP description was developed for each architecture under study. This description was used to generate a simulator for each architecture. The test programmers used the simulation facility to debug and test their programs. A set of test data was supplied for each of the test programs. A program was defined tc be debugged when it properly executed using the test data. This provided reasonable assurance of the applicability of the measures obtained without requiring proofs of the correctness of each program. A subset of the test data was used for evaluation of execution efficiency. The ISP simulator facility maintains counters of memory accesses as well as frequency of execution of each part of the simulator. These counters provided the execution statistics used in computing the architecture measures.

2.4. Measures of an Architecture's Performance

The performance of an architecture on the test programs is measured by the relative efficiency of the test programs written for that particular architecture. An efficient architecture is one which requires a small amount of storage for a test program and executes the test program in a short amount of time. Three types of measures were developed to capture this concept. The S measure is a measure of the storage requirements for a program. The M and R measures are measures of execution efficiency. See Figure 2.

2.4.1. Program Size Measure

The S measure is defined as the number of bytes of memory required by the test program. This includes stack and local variables but excludes any parameters passed and any global data structures accessed by the test program.

2.4.2. Execution Efficiency Measures

The time required for a computer to execute a given program is clearly dependent upon the hardware implementation of the machine. An arbitrary architecture may be implemented to execute its instructions more or less quickly dependent upon the technology used. The execution time of a program is determined by two factors; the amount of processing required, and the rate at which the processing is done. The rate of processing is determined by the hardware implementation. However, the amount of processing is dependent upon the program and the architecture. An efficient architecture will minimize the processing required thereby allowing the most efficient implementations. Two factors contribute to the processing required by a program: memory activity and processor activity.

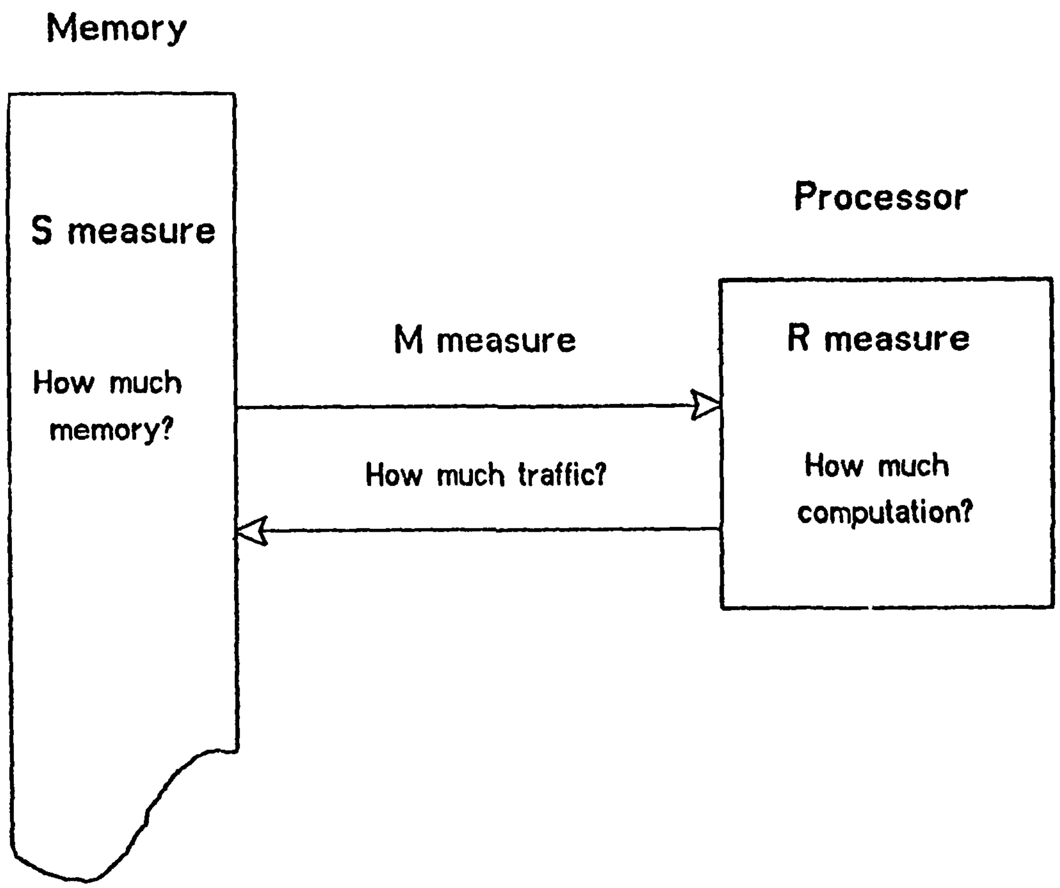
The M and R measures defined below were developed to measure those aspects of a computer architecture that would most directly affect the performance of its implementations.

<u>M- Measure of Memory Activity</u>

The most direct measure of memory activity is a count of the number of bytes read from or written into the main memory during the execution of a test program. This is precisely the definition of the M measure. In order to understand this measure, consider two different architectures which are implemented with the same processor/ memory bandwidth and memory speed. The architecture with the higher M measure for a test program would take more time to move the bytes necessary to execute the test program.

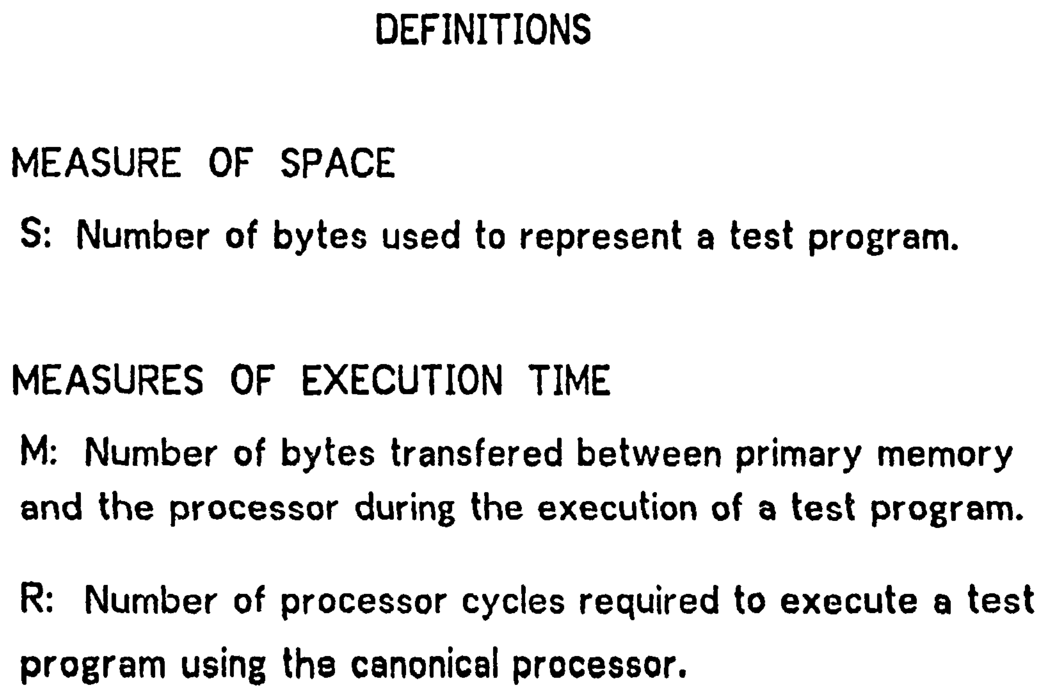
One M measure was computed for the first study. In the second

Measures of Performance of an Architecture



S is a measure of the efficiency with which a program can be represented, M and R are measures of the execution time of such a program.

Figure 2: The S, M and R measures



DEFINITIONS

MEASURE OF SPACE

S: Number of bytes used to represent a test program.

MEASURES OF EXECUTION TIME

M: Number of bytes transfered between primary memory and the processor during the execution of a test program.

R: Number of processor cycles required to execute a test program using the canonical processor.

Figure 3: Definitions of S, M and R

study, three M measures were computed to reflect the use of an 8 bit bus, a 16 bit bus, and a 32 bit bus.

R- Measure of Processor Activity

The R measure for a program is defined as the sum of the R measures for each instruction executed. The R measure of an instruction is defined as the number of processor cycles required to execute it using a canonical processor which was chosen to be representative of a typical medium performance implementation. See Figure 4. For simple instructions, the microcode was generated and the R measure was calculated directly. For more complex instructions such as integer multiply and divide and floating point operations, the R measure was established by a survey of implementation on current computers. Relative execution times were used to establish an approximate number of processor cycles required to execute the instruction. This computation cost was then added to an operand fetch cost and an instruction fetch cost to determine the R measure for the instruction.

2.5. The Statistical Design

The method used to design and analyze the experiments was based on the analysis of Variance (ANOVA) technique. ANOVA is a statistical technique for analyzing measurements which depend on several kinds of effects simultaneously, to decide which kinds of effects are important and to estimate these effects. This technique was used to identify the significant factors which influence the S, M, and R measures with emphasis on the *significance of the architecture factor*. Quantitative measures were associated with each architecture and confidence intervals at the .05 level of statistical significance were obtained for these measures. These were used to obtain statistically valid rankings of the architectures.

2.6. The Results

The results of the first study can be summarized as follows (see figure 5):

- The Interdata 8/32 architecture is significantly better than the IBM S/370 for the set of 12 test programs across all three measures of performance.

- Differences in performance between the PDP-11 and the Interdata 8/32 detected in this study are not statistically significant.

- While the PDP-11 is not shown to be superior to the IBM S/370 architecture at the .05 level, the data is skewed strongly in favor of the PDP-11.

The results of the second evaluation are summarized in figure 6. Looking at the S measure we find the 16 bit architectures make up the best group, with the PDP-11 significantly better than the AN/UYK-19. The AN/GYK-12 and AN/UYK-7, in that order, make up the worst group. In

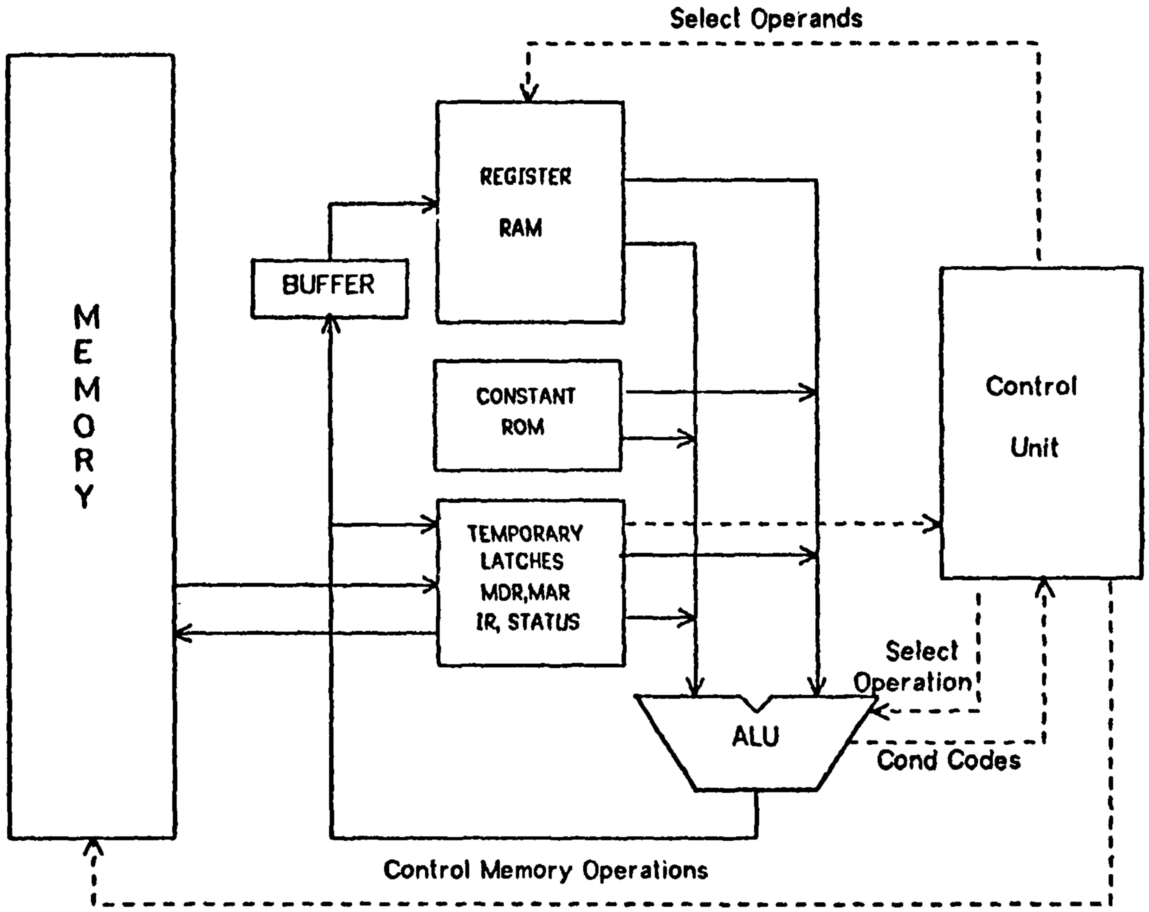


Figure 4: Canonical Processor

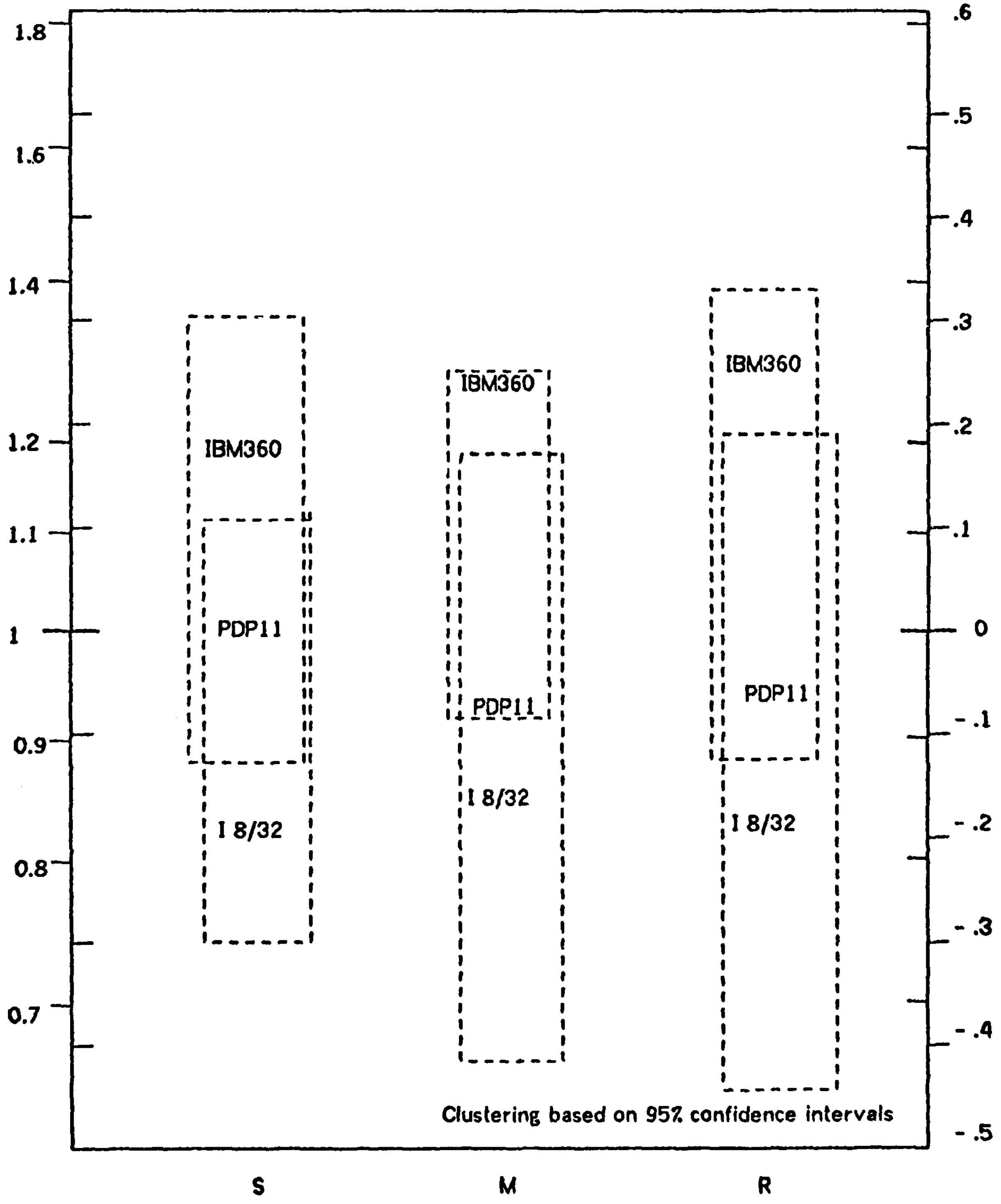


Figure 5: Results of the First Study

the M measure the AN/UYK-20 and the AN/GYK-12 both move up in the rankings relative to the others. The PDP-11 drops significantly behind the AN/UYK-20 and the AN/UYK-19 drops into the last group with the AN/UYK-7. The R measure shows the AN/UYK-20 and AN/GYK-12 clustered at the top.

While the full results for the third study are not ready at this time, it is clear from the preliminary analysis that the architectures which are similar between the second and third studies do not change their relative rankings.

3. Conclusions

In performing the three computer architecture evaluations we have devised and improved a methodology which allows the relative ranking of computer architectures based on the efficiency measures S, M and R. However, the architectures being studied in these evaluations comprise a very narrow subset of all computer architectures and the evaluations were very costly. While we have learned a great deal about comparing computer architectures we have a long way to go before we have a practical, inexpensive and effective method for evaluating computer architectures.

REFERENCES

1. Burr, William E. and Samuel H. Fuller et. al., Computer Family Architecture Selection Committee Final Report, Volume III - Evaluation of Computer Architecture via Test Programs, ECOM Research and Development Technical Report # 4528, September 1977

2. Fuller, Samuel H. and William E. Burr, Measurement and Evaluation of Alternative Computer Architectures, Computer, Volume 10 Number 10, October 1977

3. Fuller, Samuel H., G. Mathew and L. Szewerenko, Phase II Comparative Evaluation of the MCF Computer Architectures, CORADCOM Research and Development Report # 79-9, July 1978

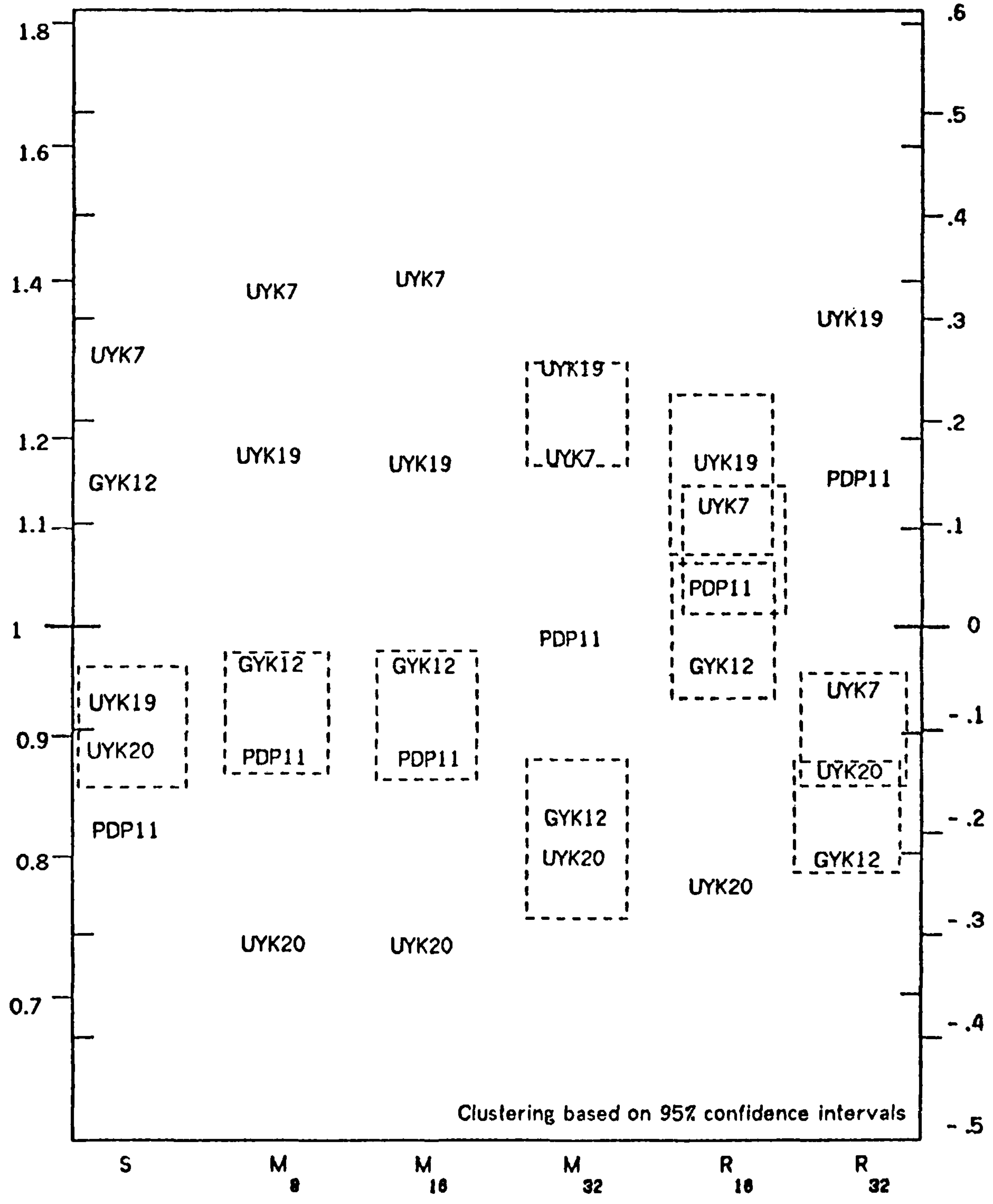


Figure 6: Results of the Second Study

RELIABILITY/SURVIVABILITY

Dan Hocking

AIRMICS

RELIABILITY/SURVIVABILITY

SESSION CHAIRPERSON: Dan Hocking

AIRMICS

SESSION SUMMARY

The reliability/survivability session consisted of a pair of presentations focused on improving the ability to demonstrate software reliability or survivability. Ray Stone of General Research Corporation presented their work on "Adaptive Testing" which is a means of automatically determining a real-time software package "performance boundary" or performance under load. Dr. Richard DeMillo, School of Information and Computer Science, Georgia Institute of Technology presented the work currently being supported under grant from AIRMICS on "Mutation Analysis" which is a means of assessing the value of test data used to verify the functions of the program. The above presentations were followed by a question and answer session led by Dan Hocking of AIRMICS.

Adaptive Testing

R. L. Stone

General Research Corporation

Adaptive Testing provides a means for automatically and efficiently determining a software package's "performance boundary". This boundary is a surface in the space of permissible input values which separates the regions of acceptable and unacceptable program performance. The Tester embodies a comprehensive way of establishing initial input values using an interactive data base construction program, data reduction programs which reduce the large quantity of performance data collected to a manageable subset and a heuristic search algorithm that determines a minimal sequence of test cases to carry out to locate the boundary. An extensive graphical aids package displays tables, graphs and surfaces on an interactive color display system under control of the user.

ADAPTIVE TESTING

R. L. Stone

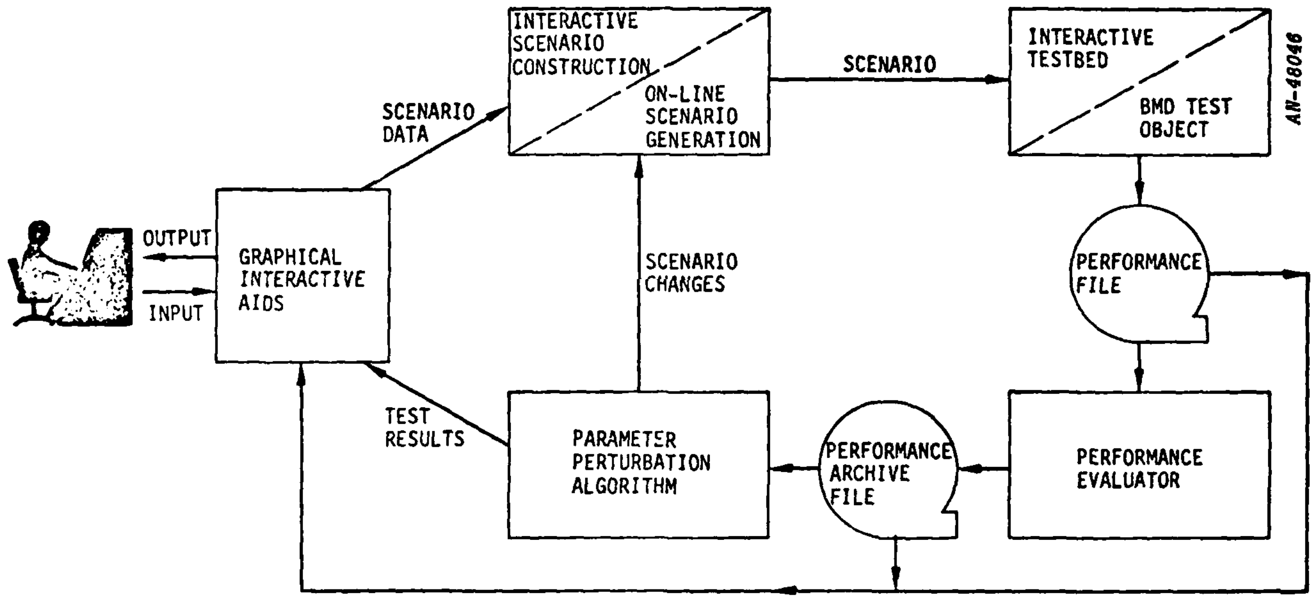
Adaptive Testing addresses the problem of locating the performance limits of data processing subsystems. Past research has been carried out in the context of Ballistic Missile Defense dp systems; the main thrust of AV&V research is to develop a tool that will identify the permissible set of attack scenarios that give rise to a high level of stress in the BMD dp system and to unacceptable performance in the total system sense.

The division in attack scenario space between those attacks that are handled successfully and those that are not is referred to as the performance boundary. The primary goal of the Adaptive Tester is to delineate the performance boundary. Adaptive techniques are used to improve the efficiency of the search process carried out over attack scenario space.

In our development of an automated test facility we have included a number of computer aids which simplify the process of assembling the initial attack scenarios, specifying the test conditions, carrying out the analysis of performance data and generating the next in the sequence of test cases. Once an adaptive test is initiated the tester proceeds in automatic closed loop mode until a scenario is found that causes the BMD test object to yield a given value for the objective function, generally the number of penetrators. The four major functions in closed loop mode are 1) generation of scenario tables from input specifications 2) execution of the simulation of the BMD process by the test object 3) evaluation of the performance of the test object and 4) perturbation of the scenario parameters by the search or Parameter Perturbation Algorithm (PPA).

ADAPTIVE TESTING

OBJECTIVE: ACCURATELY DETERMINE SOFTWARE PERFORMANCE BOUNDS AND SIGNIFICANTLY REDUCE COST AND TIME FOR TESTING THROUGH AUTOMATED AND ADAPTIVE ANALYSIS.



OUTPUT

INPUT

GRAPHICAL INTERACTIVE AIDS

SCENARIO DATA

INTERACTIVE SCENARIO CONSTRUCTION

ON-LINE SCENARIO GENERATION

SCENARIO

INTERACTIVE TESTBED

BMD TEST OBJECT

AN-48046

PERFORMANCE FILE

PERFORMANCE EVALUATOR

PERFORMANCE ARCHIVE FILE

PARAMETER PERTURBATION ALGORITHM

SCENARIO CHANGES

TEST RESULTS

RATIONALE:

- THE MAGNITUDE OF EVEN AN OPTIMAL SET OF TEST CASES RENDERS "EXHAUSTIVE TESTING" UNACHIEVABLE
- NO ORGANIZED ATTEMPTS HAVE BEEN MADE TO DETERMINE, EXPRESS IN CLOSED FORM AND UNDERSTAND THE CAUSES OF SOFTWARE PERFORMANCE LIMITS.

GRC

Our work, carried out for the data processing directorate of BMDATC, can be broken into 6 main categories.

1. The Automated Scenario generator simplifies the often arduous task of assembling the many parameters needed to complete the definition of an attack scenario. This sub-system to the Adaptive Tester provides for a global data base of threat information (complete scenario booster descriptions, object definitions) to ease the attack scenario assembly process. The program can be used with any test object since data definition templates are also maintained on the global data base.

2. Test Objects have been built according to the objectives of the research. A large effort has gone into developing test objects suitable for deterministic system-level, stochastic and algorithmic-level testing.

3. Performance evaluation research has addressed the problem of identifying those recorded parameters which can be used to best determine the next test case. Redundancy analysis, correlation, factor analysis, clustering, surface fitting have all been successfully used.

4. The heart of the tester is the adaptive search algorithm. This process decides the next test case to use based on observed past performance and a set of heuristics, either manually or automatically derived. A number of search techniques have been tried in the tester and a later slide shows their relative merit.

5. The proper use of the tester is a major concern of ours. We like to think of the tester being available during a hierarchical software development cycle. Questions arising here concern the applicability of heuristics generated for use at one level, for use with more detailed software; what is the meaning of the term "performance boundary" if the test object embodies stochastic processes; how good and complete must heuristics be to enable the tester to efficiently find a performance boundary.

6. The Adaptive tester can readily be turned into a system design aid when the input parameters are held fixed and system definition parameters modified to find an optimal configuration.

CATEGORIES OF ADAPTIVE TESTING WORK

1. AUTOMATED SCENARIO GENERATION
2. ADAPTIVE TESTER TEST OBJECTS
3. TEST OBJECT PERFORMANCE EVALUATION
4. ADAPTIVE SEARCH ALGORITHM
5. ADAPTIVE TESTING METHODOLOGY
6. OTHER APPLICATIONS OF ADAPTIVE TESTING CONCEPTS

GRC

Of course, our most concentrated efforts have gone into devising a successful, efficient adaptive search technique, for without this, adaptive testing has no chance of achieving its goals. The search algorithm must transform its view of system stress into a set of perturbations of input parameters designed to increase stress even more.

A serious research issue revolves around the impact of stochastic threat, environment or system effects on adaptive testing methodology. With a stochastic model the boundary is no longer a simple surface but must be defined in probabilistic terms. Also a number of simulation runs must be made before "average" performance against a single scenario can be ascertained.

A less esoteric but still important research issue is concerned with the design of the human interface so that maximal information about tester operation, test object behavior and test progress can be obtained with a minimum of user input.

Also, the reduction of massive recorded performance data to the minimal amount needed by the search algorithm has been a major study of the performance evaluation task. The methcdology surrounding use of the tester is of equal importance to the details of the tester itself.

ADAPTIVE TESTING

PRINCIPAL RESEARCH ISSUES

- EFFICIENT ADAPTIVE SEARCH ALGORITHM WITH LEARNING CAPABILITY
- DETERMINING DEGREE OF SYSTEM STRESS FROM PERFORMANCE DATA AND RELATING PARAMETER PERTURBATIONS TO STRESS
- IMPACT OF STOCHASTIC (THREAT, ENVIRONMENT, SYSTEM) EFFECTS ON DEFINITION OF PERFORMANCE BOUNDARY
- DESIGN AND DEMONSTRATION OF EFFECTIVE ADAPTIVE TEST TOOL WITH FLEXIBLE MAN-MACHINE INTERFACE
- HANDLING OF LARGE QUANTITIES OF PERFORMANCE DATA
- USE OF ADAPTIVE TESTER DURING HIERARCHICAL SOFTWARE DEVELOPMENT CYCLE

Our Adaptive Testing research has been carried out according to a four-year plan of which we are currently finishing the third year. During the first year we concentrated on proving the feasibility of adaptive testing against deterministic high level system-level models. In the second year we addressed the major problems of testing stochastic models: how much testing does one need to establish that a given scenario point lies on the performance boundary, and how can one reduce the number of engagement runs needed to approach the boundary.

During the third year we are testing a detailed BMD algorithm (a midcourse optical discrimination process) running on a (simulated) data processing system. We hope that we will be able to show how to "break" a dp system and thus cause the BMD system to fail in its defense objectives. In year four we intend to test some portion of real-time BMD software and demonstrate the utility of adaptive testing in determining the performance limits of a "finished" product.

AV&V FOUR YEAR PLAN

YEAR 1

- *IDENTIFY COMPONENTS OF ADAPTIVE TESTER*
- DEVELOP PROTOTYPE ADAPTIVE TESTER AND DEMONSTRATE ADAPTIVE TESTING OF DETERMINISTIC TEST OBJECTS
- DEMONSTRATE MACHINE AIDED SCENARIO CONSTRUCTION
- INVESTIGATE METHODS FOR REDUCING PERFORMANCE DATA

YEAR 2

- DEVELOP AND TEST STOCHASTIC BMD MODEL(S), SYSTEM LEVEL
- DETERMINE EFFECTS OF STOCHASTIC PARAMETERS ON PERFORMANCE BOUNDARY
- SELECT AND REFINE MOST EFFICIENT SEARCH ALGORITHM
- DEVISE METHODS FOR AUTOMATING GENERATION OF HEURISTICS
- INVESTIGATE USE OF ADAPTIVE TESTER FOR VALIDATING PERFORMANCE REQUIREMENTS

YEAR 3

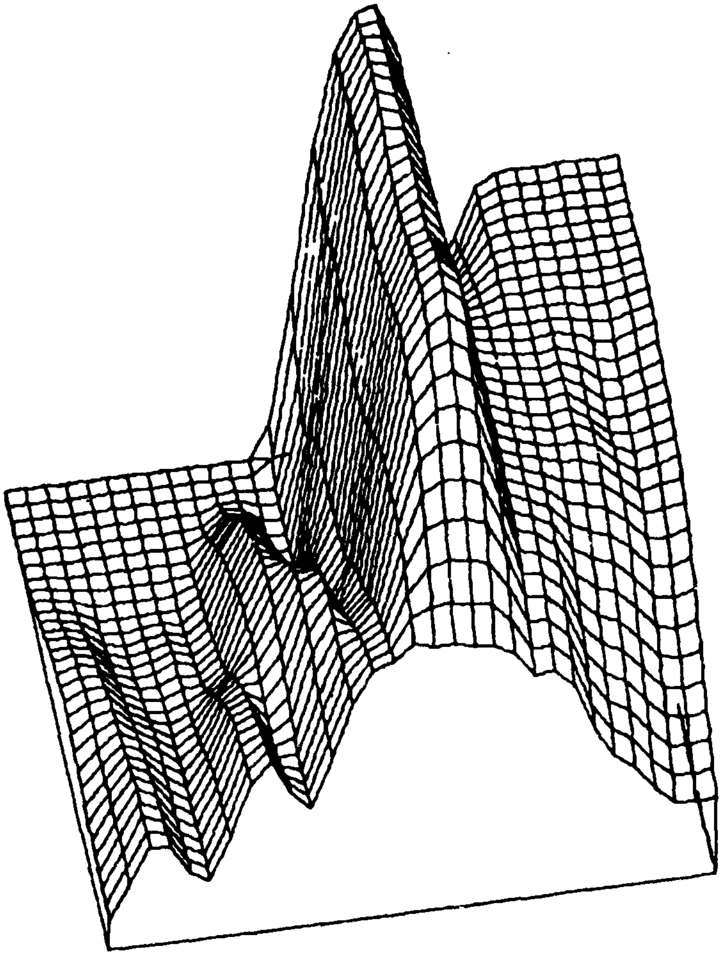
- DEMONSTRATE TESTING OF ALGORITHMIC-LEVEL BMD TEST OBJECT WHICH INCLUDES DP MODEL
- ADAPTIVELY TEST ACALP PROGRAM
- DEVELOP TECHNIQUES FOR AUTOMATIC SELECTION OF INITIAL TEST POINT IN A TEST SEQUENCE
- INVESTIGATE TRANSFERRABILITY PROPERTY OF HEURISTICS
- PREPARE FOR TESTING OF RTSW MODULE(S)

YEAR 4

- DEMONSTRATE TESTING OF RTSW
- INVESTIGATE TECHNIQUES FOR LOCATING SMALL REGIONS OF UNACCEPTABLE PERFORMANCE ("HOLES") WITHIN PERFORMANCE BOUNDARY
- FORMALIZE ADAPTIVE TESTING METHODOLOGY
- DEMONSTRATE USE OF ADAPTIVE TESTER AS A DESIGN AID

This graph shows an example of a performance surface of one of our BMD test objects. The X-direction represents the re-entry time of the first light decoy in a near-pancake attack and the Y dimension the re-entry time of the first RV. The Z-axis shows leakage (number of RV penetrations) ranging from 0-6. The remaining 7 scenario parameters were held constant (numbers of RVs, heavy decoys, light decoys; relative spacing of RVs, heavy decoys, light decoys, re-entry time of first heavy decoy.)

TO-1M PERFORMANCE SURFACE



X : Re-entry time of first light decoy in attack

Y : Re-entry time of first RV in attack

Z : Number of penetrators (Max = 6)

To be a good search algorithm (or PPA, Parameter Perturbation Algorithm) the method must be 1) Successful in that it reaches the boundary most of the time; 2) Efficient, only a small number of tests should be required; 3) Proximate, a boundary point when reached should be near the starting point; 4) Immune, not sensitive to performance surface granularities; 5) Global, global extrema are located rather than local.

We researched the properties of 4 candidate algorithms and concluded that heuristic search performed the best and had the greatest potential for overcoming its deficiencies. In the table opposite a grade of 1 indicates the best, 4 the worst.

PARAMETER PERTURBATION ALGORITHM RESULTS

PPA TYPE PROPERTY	GRADIENT	RANDOM	COMPLEX	HEURISTIC
SUCCESSFUL	3	1	2	2
EFFICIENT	1	2	3	1
PROXIMATE	2	4	3	1
IMMUNE	4	1	3	2
GLOBAL	4	1	2	3

- RANDOM AND COMPLEX SEARCH PROVED BEST WHEN LITTLE WAS KNOWN ABOUT THE PERFORMANCE CHARACTERISTICS OF THE TEST OBJECT

- GRADIENT SEARCH PERFORMANCE IS STRONGLY DEPENDENT ON THE REGULARITY OF THE PERFORMANCE SURFACE

- HEURISTIC SEARCH WAS MOST EFFICIENT AND SUCCESSFUL

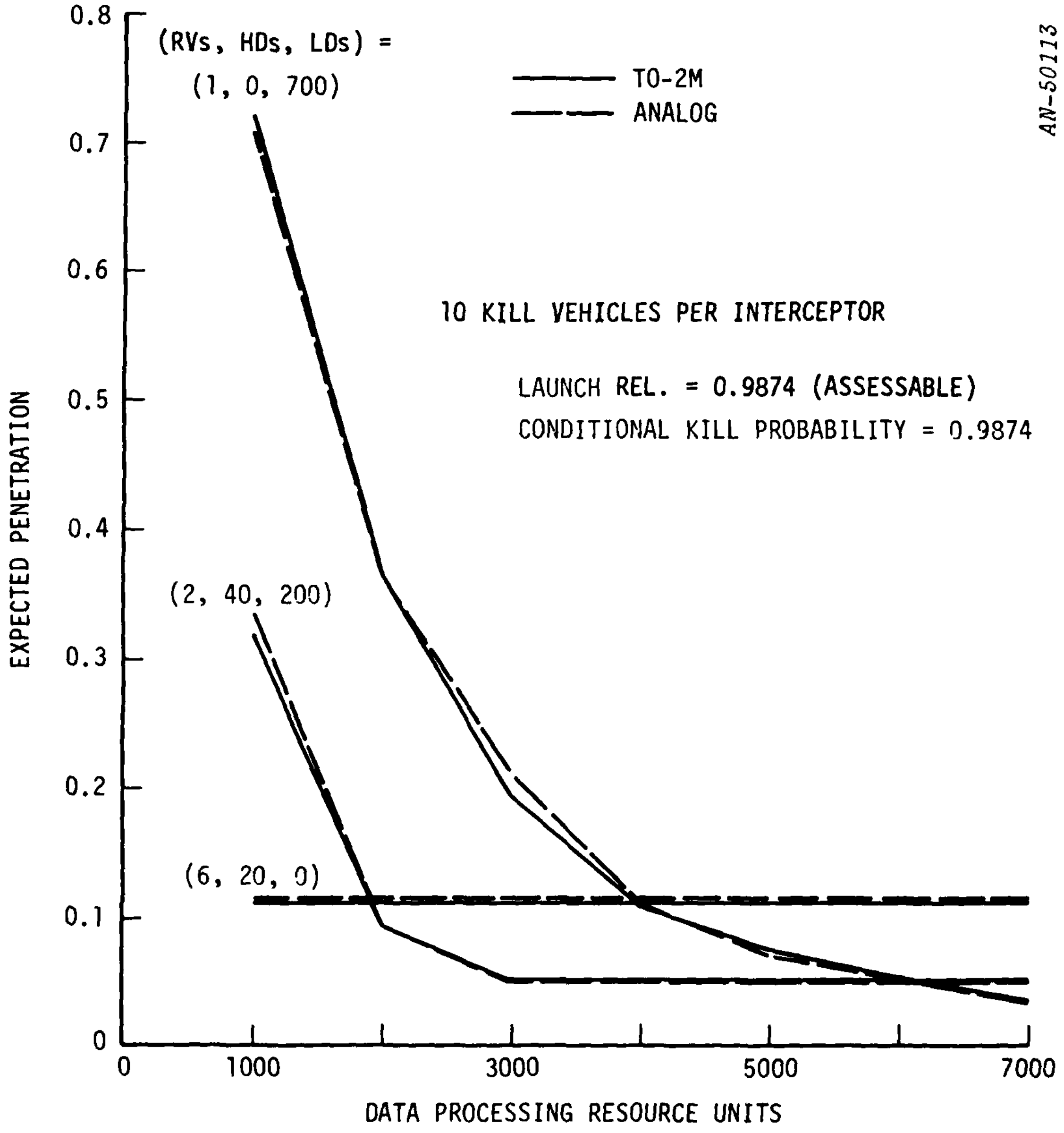
- HEURISTIC SEARCH LACKS "GLOBAL" PROPERTY

- HEURISTIC SEARCH PERFORMANCE IS STRONGLY DEPENDENT ON "GOOD" HEURISTICS (COMPLETE AND CORRECT)

- NO ONE SEARCH TECHNIQUE PROVED TO BE BEST IN ALL CASES

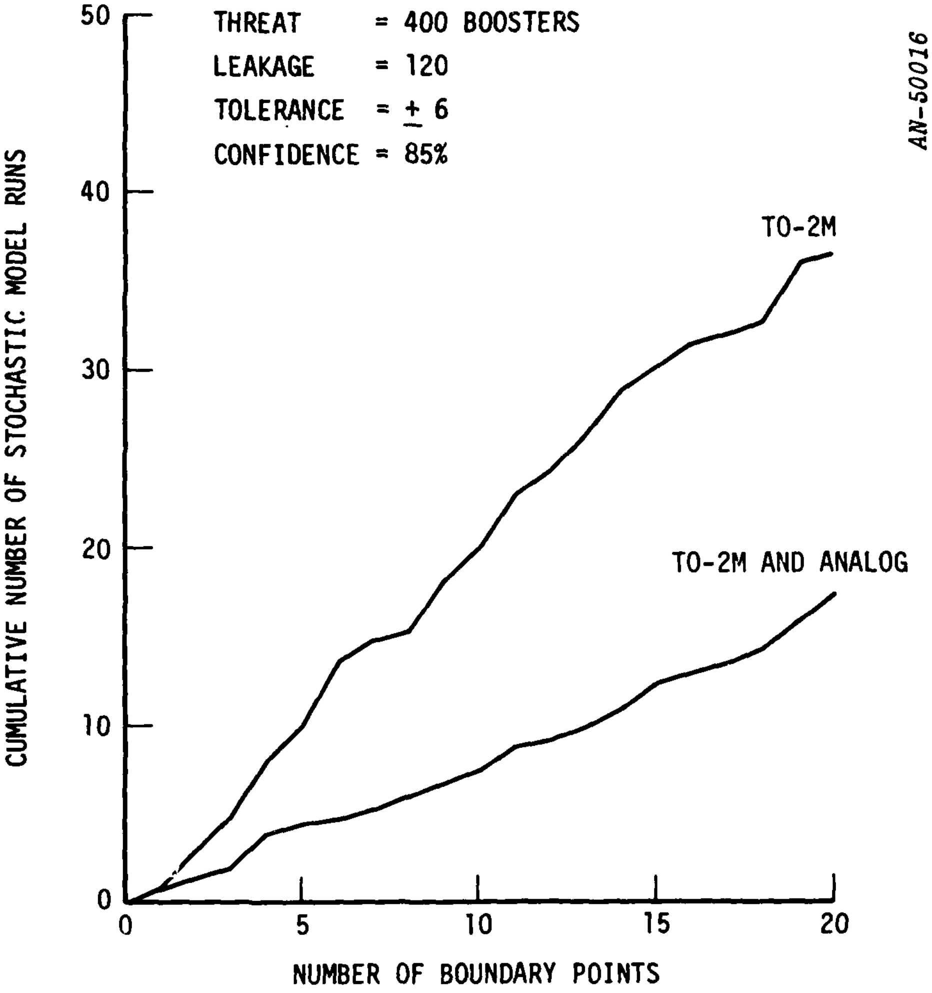
We built an expected value analog to our TO-2M stochastic test object to demonstrate the cost reduction possible in testing stochastic models if such a deterministic model were available. The next two slides show the agreement between the expected value analog and its stochastic model and the effectiveness of using such an analog in testing stochastic models. The reduction in the number of test runs required of the stochastic model dropped dramatically when the expected value analog was used to approach the boundary before the stochastic model was used to verify that the boundary had indeed been reached.

COMPARISON OF EXPECTED PENETRATION IN TO-2M AND ITS ANALOG, CASE 1



GRC

EFFECTIVNESS OF TO-2M ANALOG



AN-50016

SUPPORTING VUGRAPHS FOR
ADAPTIVE TESTING PRESENTATION

GRC

ADAPTIVE TESTING RATIONALE

TESTING SHOULD BE:

● EFFICIENT:	MINIMIZE COST MEET SCHEDULE
● INFORMATIVE:	FIND WHICH ALLOWED INPUTS (THREATS) CAUSE FAILURE LOCALIZE DESIGN DEFICIENCIES

TEST OBJECTIVES SHOULD BE CAREFULLY DEFINED:

- "100% GUARANTEES" INFEASIBLE FOR LARGE PROGRAMS
- TEST EFFORT DEPENDS STRONGLY ON NATURE OF OBJECTIVES AND CONFIDENCE LEVELS DEMANDED

ADAPTIVE VERSUS HUMAN TESTING

- ADAPTIVE TESTER IMPLEMENTS A FULLY SPECIFIED TEST PROCEDURE MUCH MORE EFFICIENTLY THAN A HUMAN
- ADAPTIVE SELECTION OF TEST CASES CAN BE EITHER BETTER OR WORSE THAN A HUMAN SELECTION (IN ANY EVENT, TESTER DOES NOT GET TIRED OR BORED)
- NO DIRECT COMPARISON OF ADAPTIVE VERSUS HUMAN TESTING FOR THE SAME TEST OBJECT IS CURRENTLY AVAILABLE

BUT

- IT IS POSSIBLE TO MEASURE IMPROVEMENTS IN ADAPTIVE TESTER CAPABILITY AND DESIGN:
  - TEST STRATEGIES AND ALGORITHMS
  - ACCESS AND DISPLAY OF ARCHIVED PERFORMANCE DATA

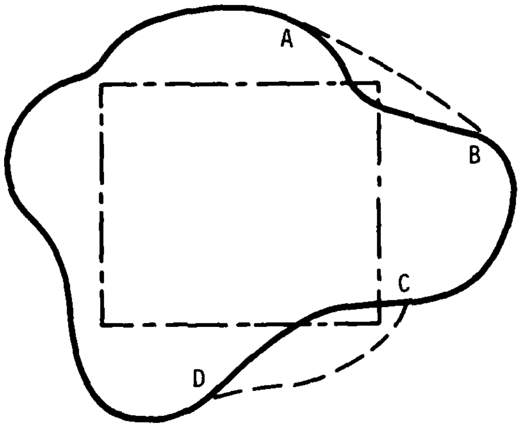
CONCEPTS AND PROBLEMS IN ADAPTIVE TESTING

1. THE TEST OBJECTIVE FUNCTION AND THE "PERFORMANCE BOUNDARY" CONCEPT

- FOR WHAT THREATS IS THE OBJECTIVE ONLY JUST MET?
  HOW DO THEY RELATE TO THE DESIGN THREATS?

- WHERE IS DP THE LIMITING FACTOR?
  HOW SEVERE IS THE LIMITATION?

CRITICAL THREATS DEFINE A SURFACE IN "THREAT SPACE." WE CALL IT THE "PERFORMANCE BOUNDARY."



━━━━ PERFORMANCE BOUNDARY OF TEST OBJECT

-·-·-·- DESIGN THREAT BOUNDARY

— — — MODIFICATIONS TO BOUNDARY WHEN DP CAPABILITIES UNLIMITED

OBSERVATIONS:

- DP IS LIMITING FACTOR FOR THREATS ON AB AND CD, BUT NOT ELSEWHERE ON THE BOUNDARY
- DP IMPROVEMENTS CAN ENABLE SYSTEM TO MEET DESIGN OBJECTIVE

CONCEPTS AND PROBLEMS IN ADAPTIVE TESTING

2. STOCHASTIC TEST OBJECTS

- STOCHASTIC EFFECTS ARE A PROBLEM FOR BOTH HUMAN AND ADAPTIVE TEST SELECTION AND EVALUATION
- TEST REPETITIONS TO ESTABLISH AVERAGE PERFORMANCE VERSUS GIVEN THREAT MUST BE SUFFICIENT TO BOTH:

  (1) ENABLE GOOD SELECTION OF NEXT TEST CASE

  (2) DETERMINE WHETHER OR NOT TEST OBJECTIVE HAS BEEN REACHED
- FOR (1):

  NUMBER OF REPETITIONS DEPENDS ON SEARCH HEURISTICS:

  - ∿50 FOR GOOD HEURISTICS SET
  - ? FOR HUMAN
- FOR (2):

  NUMBER OF REPETITIONS DEPENDS ON:

  - PERFORMANCE BOUNDARY DEFINITION; E.G., EXPECTED VALUE OF OBJECTIVE FUNCTION LIES IN

    $$\vec{V} - 1/2\,\vec{\delta}\ ,\ \vec{V} + 1/2\,\vec{\delta}$$

    WITH C% CONFIDENCE
  - RATIO OF STANDARD DEVIATION TO MEAN VALUE OF OBJECTIVE FUNCTION AT THE TEST POINT

CONCEPTS AND PROBLEMS IN ADAPTIVE TESTING

3. APPROACHES TO TESTING STOCHASTIC MODELS

- IF POSSIBLE, DEVELOP A DETERMINISTIC MODEL ABLE TO GIVE A GOOD FORECAST OF EXPECTED VALUES OF THE STOCHASTIC OBJECTIVE FUNCTION ("DETERMINISTIC ANALOG"):
  - MINIMIZES OFF-BOUNDARY TESTING
  - DOES NOT AFFECT BOUNDARY POINT CONFIRMATION TESTING
- IMPROVE HEURISTIC SET:
  - REDUCES REPETITIONS PER TEST POINT AT NON-BOUNDARY POINTS, AND NUMBER OF TEST POINTS NEEDED TO REACH BOUNDARY
  - DOES NOT AFFECT BOUNDARY POINT CONFIRMATION TESTING

CONCEPTS AND PROBLEMS IN ADAPTIVE TESTING

4. SOURCES OF HEURISTICS

- DESIGNER KNOWLEDGE OR BELIEF ABOUT TEST OBJECT BEHAVIOR
- PATTERNS OF TEST OBJECT BEHAVIOR REVEALED BY AUTOMATED PROCESSING OF PERFORMANCE DATA. TECHNIQUES INCLUDE:
  - DIMENSION REDUCTION
  - CLUSTER FORMATION

  "STRESS MEASURES" ARE PRODUCED.

  NOTE: THIS SUBSTITUTES FOR HUMAN ASSESSMENT OF "WHAT TO DO NEXT"

  STATUS. TECHNIQUES HAVE BEEN IMPLEMENTED TO BE APPLIED TO TO-3, WHICH INCLUDES DETAILED DP MODEL

MAJOR TECHNICAL QUESTIONS

1. CAN WE DERIVE GOOD SETS OF SEARCH HEURISTICS FOR BMD SYSTEMS AT ALL LEVELS OF REPRESENTATION? TO WHAT EXTENT ARE THE HEURISTICS TRANSFERABLE:

   - FROM ONE LEVEL TO THE NEXT
   - BETWEEN SIMILAR BMD CONSTRUCTS

2. (a) CAN STRESS MEASURES BE DEVELOPED THAT RELATE DIRECTLY TO THOSE PARTS OF THE PERFORMANCE BOUNDARY WHERE DP IS THE LIMITING FACTOR?

   (b) DO THESE STRESS MEASURES INDICATE WHY DP IS LIMITING?

3. HOW CAN WE ESTABLISH THAT THERE ARE NO "HOLES" INSIDE A NOMINAL PERFORMANCE BOUNDARY?

A HEURISTIC IN PRODUCTION RULE FORM

- JUSTIFICATION ⟶ ACTION ⟶ RESULT

JUSTIFICATION TELLS UNDER WHAT CONDITIONS THE HEURISTIC CAN BE USED

ACTION TELLS HOW THE THREAT VARIABLES ARE TO BE PERTURBED

RESULT TELLS WHAT KIND OF CHANGE IN PERFORMANCE IS EXPECTED

EXAMPLE OF TO-1M HEURISTIC TEST SET

NO. OF HEURISTICS	7
HEURISTIC	J1: A1
"	J2: A1, A2
"	J2: A3, A4
"	J3: A3, A5
"	J4: A6
"	J5, J6: A5, A6
"	J7: A7, A8, A9
NO. OF JUSTIFICATIONS	7
JUSTIFICATION	J1: I3/I1 .GT. 400
"	J2: I3/I1 .LE. 400
"	J3: I2/I1 .LE. 20/6.
"	J4: I2/I1 .GE. 30
"	J5: I2/I1 .LT. 30
"	J6: I2/I1 .GT. 20/6.
"	J7: .TRUE.
NO. OF ACTIONS	9
ACTION	A1: I3 = I3 + 50
"	A2: I2 = I2 - 5
"	A3: I1 = I1 - 1
"	A4: I3 = I3 + 100
"	A5: I2 = I2 + 10
"	A6: I1 = I1 + 1
"	A7: I1 = I1 + RNV(1,1)
"	A8: I2 = I2 + RNV(10,10)
"	A9: I3 = I3 + RNV(100,100)

GRC

AD-A104 252   INTERNATIONAL BUSINESS SERVICES INC WASHINGTON DC   F/G 9/2
US ARMY SOFTWARE SYMPOSIUM (2ND) HELD AT WILLIAMSBURG, VIRGINIA--ETC(U)
1978   S M TAYLOR   DAAK70-78-D-0030

UNCLASSIFIED   NL

10 OF 10
AD
A104252

END
DATE
FILMED
10-81
DTIC

AV&V ONGOING ISSUES

- AVAILABILITY OF TEST OBJECTS
- COST OF RUNNING TEST OBJECTS
- QUANTIFYING PAYOFFS OF ADAPTIVE TESTING
- GRAPHICAL, FUNCTIONAL REPRESENTATION OF PERFORMANCE BOUNDARY
- DP MODELS SUFFICIENTLY DETAILED AND REALISTIC TO PRODUCE MEANINGFUL PERFORMANCE MEASURES
- SHOWING TRANSFERRABILITY PROPERTY OF HEURISTICS
- DEALING WITH HIGH DIMENSIONALITY IN THREAT SPACE

GRC

ADAPTIVE VERIFICATION AND VALIDATION DOCUMENTS

NUMBER	TITLE	AUTHOR	DATE
CR-1-708	RESEARCH PLAN FOR ADAPTIVE V&V	E. BULEY, et al.	5/76
CR-2-708	ADAPTIVE V&V RESEARCH EVALUATION PLAN	E. BULEY, et al.	6/76
CR-3-708	AUTOMATED SCENARIO GENERATOR, SOFTWARE DESIGN DOCUMENT	J. LINDER, R. STONE	7/76
CR-4-708	ADAPTIVE LEARNING REQUIREMENTS AND CRITICAL ISSUES	D. COOPER	1/77
CR-5-708	ADAPTIVE V&V FINAL REPORT	E. BULEY, et al.	1/77
CR-1-767	REQUIREMENTS DEFINITION FOR THE TEST OBJECT 2 MODEL	R. UTTLEY	3/77
CR-2-767	AUTOMATED SCENARIO GENERATOR, SOFTWARE TEST DEFINITION	R. STONE	2/77
CR-3-767	ASG ACCEPTANCE TEST REPORT	R. STONE	6/77
CR-4-767	AV&V RESEARCH PLAN UPDATE	R. UTTLEY, D. COOPER	6/77
CR-5-767	ISC USER'S MANUAL	R. STONE	6/77
CR-6-767	ADAPTIVE V&V MID-YEAR REPORT	R. UTTLEY, et al.	7/77
CR-7-767	ADAPTIVE V&V RESEARCH EVALUATION PLAN UPDATE	R. UTTLEY	9/77
CR-8-767	AV&V RESEARCH PROGRESS EVALUATION REPORT	R. STONE, et al.	12/77
CR-9-767	AV&V FINAL REPORT-II	R. STONE, et al.	1/78

NUMBER	TITLE	AUTHOR	DATE
CR-1-826	TO-3 ALGORITHM LEVEL TEST OBJECT DEFINITION	H. F. GILMORE, et. al.	7/78
CR-2-826	PERFORMANCE MEASURES VALIDATION TECHNIQUES	E. R. BULEY	9/78
CR-3-826	ADAPTIVE TESTER DEMONSTRATION PLAN	R. STONE	10/78
CR-4-·826	ADAPTIVE TESTING METHODOLOGY	R. UTTLEY	11/78
CR-5-826	ADAPTIVE V&V FINAL REPORT YEAR 3	R. STONE, et. al	1/79

GRC

APPENDIX

US ARMY SECOND SOFTWARE SYMPOSIUM

ATTENDEE LIST

Mack Alford
TRW
7702 W. Governors Drive
Huntsville, ALA. 35805

Serafino Amoroso
CORADCOM
ATTN: TCS-BG
Ft. Monmouth, N.J. 07703

Paul Applin
SDC
P.O. Box 157
Ft. Monroe, VA 23651

Wallace C. Arnold
ODCSAC
H&PA
DAAC-SIF
Washington, D.C. 20310

George Ashendorf
TRI-TAC
VomTTacTical Communications
Office
TT-LD-ILS
Ft. Monmouth, N.J. 07703

J. C. Ashlock
JPL
4800 Oak Grove Dr.
Pasadena, CALIF. 91103

Steven A. Austin
US Army
COARDCOM
CDR, CORADCOM
ATTN: DRDCO-SEI-I
Ft. Monmouth, N.J. 07703

Bob Barrier
Raven Systems & Research
225 Peachtree Street
Suite 225
Atlanta, Georgia 30303

Victor Basili
University of Maryland
Dept. of Computer Science
College Park, MD 20742

Merton J. Batchelder
USACSC-CSCS-POP
Ft. Belvoir, VA 22060

Edward J. Beach
CENTACS, CORADCOM
DRDCO-TCS-BK
Ft. Monmouth, N.J. 07703

Howard F. Bleakney
HQ-USACC
HQ US Communications Command
AMG-MISR
Ft. Huachuca, AZ 85613

E. G. Bourlas
USACSC
Fort Belvoir, VA 22060

Jerry R. Brookshire
USAMIRADCOM
DRMI-TGG
Redstone Arsenal, AL 35809

R. Peyton Brown
USACSC
Ft. Belvoir, VA 22060

MG. Clay T. Buckingham
Commander, U.S. Army Computer
Systems Command
Washington, D.C. 20310

William E. Burr
CENTACS
Institute for Computer Science
& Technology
National Bureau of Standards
Washington, D.C. 20234

LTC Roy Busdiecker
HQDA
HQDA (DAAC-PE)
Washington, D.C. 20310

Leslie G. Callahan Jr.
GA. Institute of Technology
Atlanta, Georgia 30332

LTC Robert P. Campbell
HQDA (DAMI-AMP)
Pentagon
Washington, D.C. 20310

John Clark
USACSC, SpGp
Ft. Lee, VA 23801

James Clary
Research Triangle Inst.
P. O. Box 12194
Research Triangle Park, NC 27709

John M. Cole
COARDCOM
Ft. Monmouth, NJ 07703

W. A. Coleman
USACSC
Ft. Belvoir, VA 22060

1LT Richard Conn
US Army Satellite Comm. Agcy
USA SATCOM A (DRCPM-SC-4G)
Ft. Monmouth, NJ 07703

A. B. Connelly
USACC
HQ USACC, CC-OPS-TC
Ft. Huachuca, AZ 85613

Robert L. Cooper
OASC (C) DDA
Room/A658
Pentagon
Washington, D.C. 20301

J. Mike Coward
Teledyne Brown
300 Sparkman Dr. MS 212
Huntsville, ALA. 35807

Allan H. Curry
AIRMICS
313 Calculator Bldg.
Georgie Institute of Technology
Atlanta, GA 30332

Fred J. D'Ascoli
USACSC
Fort Belvoir, VA 22060

Joseph W. D'Oria
CORADCOM
ATTN: DRDCO-PT
Ft. Monmouth, NJ 07703

Dr. Thomas G. DeLutis
The Ohio State University
Dept. of Computer & Info. Sci.
2036 Neil Avenue Mall
Columbus, Ohio 43210

Richard DeMillo
Georgie Institute of Technology
Dept. of Computer Science
Atlanta, GA 30332

Mr. Barry DeRoze
TRW Systems Group
1 Space Park
Bldg R 2/1086
Redondo Beach, CA 90278

Thomas Dames
CORADCOM
DRDCO-AM
Ft. Monmouth, NJ 07793

Carl G. Davis
BMDATC
P. O. Box 1500
Huntsville, ALA 35803
ATTN: ATC-P

J. G. Demko
DMIS-CERCOM
Ft. Monmouth, NJ 07703

William Dietz
Carnegie-Mellon University
Computer Science Dept.
Pittsburgh, PA 15213

Anthony Digiorgio
CORADCOM
CDR, CORADCOM
ATTN: DRDCO-SEI-I
Ft. Monmouth, NJ 07703

Timothy A. Dreisbach
SofTech
460 Totten Pond Rd.
Waltham, MA 02154

Lorraine Duvall
IITRI
P. O. Box 1355 Branch P. O.
Rome, N. Y. 13440

David Egli
CORADCOM
CENTACS
Ft. Monmouth, NJ 07703

Ingrid Eldridge
CENTACS
COARDCOM DRDCO-TCS-BK
Ft. Monmouth, NJ 07703

Edward H. Ely
AIRMICS
313 Calculator Bldg.
Georgia Institute of Technology
Atlanta, GA 30332

CPT Edward Errickson
USAICS
P. O. Box 11472
Phoenix, AZ. 85061

Bill Fallon
HQTRADOC (ATCD-C-D)
Ft. Monroe, VA 23651

Gilbert M. Fariss
USACSC
US Army Computer Systems CMD.
ATTN: CSCS-ACC (STOP 110)
Ft. Belvoir, VA 22060

Kurt Fischer
CSC
6565 Arlington Blvd.
Falls Church, VA 22046

W. C. Frey
JPL
4800 Oak Grove Dr.
Pasadena, CALIF 91103

J. A. Garretson
McDonnel Douglas
5301 Bolsa Avenue
Huntington Beach, CALIF 92647
MS 11-3

James W. Gault
NC State University
Dept. of EE, NCSU
Raleigh, N. C. 27650

Susan Gerhart
USA/ISI
4676 Admiralty Way
Marina del Ray, CALIF 90291

Russell Green, Dir.
USAMSSA
ATTN: ACAM-DPD-E-BD969
Pentagon
Washington, D.C. 20301

H. Mark Grove
OUSA (R&E) AP
Rm. 2A318
Pentagon
Washington, D.C. 20301

Margaret Hamilton
Higher Order Software Inc.
806 Mass. Avenue
Cambridge, MA 02139

Harry F. Hardin
US Army Corps of Engrs.
HQDA (DAEN-CWE-BA)
1000 Independence Avenue
Forrestal Bldg. Rm. 5H088
Washington, D.C. 20314

James H. Herd
Doty Associates
416 Hungerford Dr.
Rockville, MD 20850

L. T. Herrmann
Shell Oil Co.
P. O. Box 20127
Houston, Texas 77025

Bertram Herzog
University Computing Center
University of Colorado
Boulder, CO 80309

B. J. Hill (PM-ARTADS, TSSG)
Box 3045
Ft. Sill, OK 73503

Edward B. Hirsch
US Army CERCOM-DRSEL-ME-SC
Ft. Monmouth, NJ 07703

Morton A. Hirschberg
US Army ARRADCOM
Ballistic Research Lab
DRDAR-BLB
Aberdeen Proving Ground, MD 21005

Carl Hitchon
SofTech
460 Totten Pond Rd.
Waltham, MA 02154

Dan Hocking
AIRMICS
313 Calculator Bldg.
Georgia Institute of Technology
Atlanta, GA 30332

Jean N. Hooper
ARI
USARI
PERI-OS
5001 Eisenhower Avenue
Alexandria, VA 22333

Pei Hsia
UAH
Computer Science Dept., USH
P. O. Box 1247
Huntsville, AL 34807

Dwain B. Huewe
CENSEI CORADCOM
DCRDO-SE
Ft. Monmouth, NJ 07703

Lee Hughey
Raven Systems & Research
225 Peachtree Street
Suite 225
Atlanta, GA 30303

Windell F. Ingram
USAE Waterways Experiment Station
P. O. Box 631
Vicksburgh, MS 39056

Atul R. Jai
Research Triangle Institute
Box 12194
Research Triangle Park, NC 27709

James R. Jancaitis
USAETL
Bldg. 2592
Ft.Belvoir, VA 22060

Medhi Jazayeri
University of NC
Computer Science Dept.
Chapel Hill, NJ 27514

Jerry R. Jeffrey
USA ADMINCEN
US Army Administration Center
ATTN: ATZI-S
Ft. Harrison, IN 46216

Jon C. Jervert
CORADCOM
CENTACS
Ft. Monmouth, NJ 07703

James M. Jones II
USAEWES
ADP Center
Waterways Experiment Station
P. O. Box 631
Vicksburg, MS 39180

Dr. Larry A. Johnson
LOGICON
18 Hartwell Avenue
Lexington, MA 02173

Philip Johnson
USA CORADCOM CENSEI
Ft. Monmouth, NJ 07703

Helmuth Kaunzinger
US Army
CORADCOOR
DRDCO-TCS-BG
Ft. Monmouth, NJ 07703

Klaus P. Koschewa
AIRMICS, GIT
313 Calculator Bldg.
Atlanta, GA 30332

R. M. Lamb
USAICS/ATSI-CD-CS
Ft. Huachuca, AZ 85613

Doug Langley
USACSC
Ft. Belvoir, VA 22060

Ed Lee
Raytheon Company
Hartwell Road
Bedford, MA 01780

Dr. E. Lieblein
CENTACS, CORADCOM
HQ CORADCOM
ATTN: DRDCO-TSC-BH
Ft. Monmouth, NJ 07703

COL James E. Love
USACSC STOP-C-170
Ft. Belvoir, VA 22060

G. A. Mackay
Commander
USACEEIA
CC-TAD-TST
Ft. Huachuca, AZ 85613

G. Scott Mackay
Commander
HQ USACESSIA, CCC-TAD-TJD
Ft. Huachuca, AZ. 85613

E. J. McCauley
Ford Aerospace & Comm. Corp.
FACC
Mail Stop V-02
3937 Fabian Way
Palo Alto, CALIF. 94303
(netmail McCauley @SRI-KL)

D. R. McClung
Patriot Project Office
DRCPM-MD-T-S
Redstone Arsenal, ALA. 35809

Robert A. McMurrer
USA Corps of Engineers
HQDA (DAEN-DSE)
Washington, D.C. 20314

MAJ Robert W. Mace
Commander
USASC & FG
ATTN: ATZHTD-P (MAJ Mace)
Fort Gordon, CA 30905

Dennis P. Mahoney
USACACDA-ATCA-BA
Ft. Leavenworth, KS. 66027

John R. Mitchell
AIRMICS
313 Calculator Bld.
Georgia Tech. 30332

Roy Mattson
CORADCOM, CENTACS
Ft. Monmouth, NJ 07703

George Mikula
HQDARCOM
5001 Eisenhower Avenue
Alexandria, VA 22333

Clair R. Miller
Honeywell
7900 West Park Drive
McLean, VA 22309

Derek S. Morris
CENTACS, CORADCOM
ATTN: TCS-BG
Ft. Monmouth, NJ 07703

MAJ Albert A. Mullin
USABMD ATC
P. O. Box 1500
Huntsville, ALA. 35808

J. David Naumann
University of Minnesota
271-19th Avenue, South
Minneapolis, MN 55455

J. Gary Nelson
TECOM, APG-Md
USA TECOM
ATTN: DRSTE-AD-S
Aberdeen PG. MD. 21015

John A. Nicholas
HQDARCOM (DRCDE-C)
5001 Eisenhower Avenue
Alexandria, VA 22333

MAJ Marlan L. Nienhuis
USACSCSPTGP
Ft. Lee, VA 23801

Donald B. Nowaskoski
Western Union
7916 Westpark Dr.
McLean, VA 22102

Edward J. O'Connor
USATCOMA
DRCPM-SC-IE
Ft. Monmouth, NJ 07703

John J. O'Hare
ONR
800 N. Quincy Street
Alexandria, VA 22217

LTC Donald E. Painter
ACSAC, HQDA
ATTN: DAAC-SIF (LTC. D. Painter)
Washington, D.C. 20310

Gary L. Peckham
Engineering Experiment Station
Georgia Institute of Technology
Atlanta, GA 30332

Sil Pelosi
CENTACS
HQ CORADCOM DRC-TDS-B
Ft. Monmouth, NJ 07703

G. Perrone
USACEEIA
CCC-SEO
Rm. 3501 Greecy Hall
Ft. Huachua, AZ. 85613

J. Petterson
USACSC
7203 Warwick Drive
Camp Springs, MD 20031

Donald L. Phillips
US Army Engr. Dist. Jacksonville
P. O. Box 4970
Jacksonville, FLA 32201

John N. Postak
Doty Associates, Inc.
416 Hungerford Drive
Rockville, MD 20850

Dr. Lawrence M. Potash
US Army Research Instutute
5001 Eisenhower Avenue
Alexandria, VA 22333

Lawrence Putnam
Quantitative Software Mgt.
1057 Waverley Way
McLean, VA 22101

Dr. N. Radhakrishnan
USAEWES
Special Technical Asst.
ADP Center
Waterways Experiment Station
P. O. Box 631
Vicksburg, MS. 39180

CAP John T. Ratzenberger
USACACDA
ATTN: ATCA-BAI-M
Ft. Leavenworth, KS 66027

B. G. Leonard Riley
Deputy Commander,
US Army Computer Systems
Command
Ft. Belvoir, VA 22060

Thomas A. Rorro
BETA JPO
BETA-SE
Harry Diamond Labs
Adelphi, MD 20783

Charles W. Rose
Case Western Reserve U.
Crawford Hall
Cleveland, OH 44106

Robert Rosen
HDL,DELHD-IR
2800 Powder Mill Road
Adelphi, MD 20783

Ingo E. Rucker
US Army
ARRADCOM/MISD B/350
Dover, N.J. 07801

Myron S. Samuel
US Army Satellite Comm. Agcy.
USASATCOMA (DRCPM-SC-4G)
Ft. Monmouth, NJ 07703

A. Saucier
DARCOM
5001 Eisenhower Avenue
Alexandria, VA 22333

Samuel H. Scalzo
Commander
USAARRCOM HQ
ATTN: DRSAR-MAG-I
Bldg #9/S. Scalzo
Dover, NJ 07801

MAJ J. H. Schroeder
USAADS/DCD
Ft. Bliss, TX 79916

Marvin Schwartz
CORADCOM CENTACS DRDCO-TCS-BK
Ft. Monmouth, NJ 07703

James E. Scott
USAMIRADCOM
DRMI-TGG
Redstone Arsenal, ALA 35809

John W. Severin
USACSC-SGL
US Army Computer System Command
Ft. Lee, VA 23894

Hugh H. Sharp, III
Huntsville Division
Corps of Engineers
P. O. Box 1600
Huntsville, ALA 35807

Alan Sherer
CSC
6022 Technology Dr.
Huntsville, ALA. 35803

LTC Joseph P. Shine
AIRMICS
313 Calculator Bldg.
Atlanta, GA 30332

Raymond C. Sidorsky
USARI
PERI-OS
5001 Eisenhower Avenue
Alexandria, VA 22314

A. P. Simkus
ARO
P. O. Box 12211
Research Triangle Park, NC 27709

Bobby B. Simmons
USACSC
Ft. Belvoir, VA 22060

Daniel G. Smith
LOGICON
18 Hartwell Avenue
Lexington, MA 02173

Wesley E. Snyder
N.C. State University
Eletrical Engineering Dept., NCSU
Raleigh, NC 27650

Wayne Spruell
USAMIRCOM
DRSMI-WSP
Redstone Arsenal, ALA. 35802

Dr. John Staudhammer
ARO
P. O. Box 12211
Research Triangle Park, NC 27709

D. W. Stewart
USACSC
Ft. Belvoir, VA 22060

Gaye Stewart
Raven Systems & Research
225 Peachtree
Suite 715
Atlanta, GA 30303

R. R. Stillwagon
HQ DA (CDAAC-SIF)
Washington, D.C. 20310

Harold Stone, Prof.
University of Massachusetts
Dept. of Electrical and Computer Eng.
Amherst, MA. 01003

Ray L. Stone
GRC
P. O. Box 6770
Santa Barbara, CALIF. 93111

Leon G. Stucki
BCS
P. O. Box 24346
Seattle, WA 98124

James E. Studer
DAMI-AM
HQDA (DAMI-AM)
Pentagon
Washington, D.C. 20310

Mary A. Tate
CENSEI
CORADCOM
DRDCO-SEI-V
Ft. Monmouth, NJ 07703

Norman J. Taupeka
CORADCOM
Ft. Monmouth, NJ 07703

Dr. Stanley M. Taylor
USA/BRL/ARRADCOM
USA Ballistic Research Lab.
Aberdeen Proving Ground, MD. 21005

Milton Tenzer
CENTACS
CORADCOM
Ft. Monmouth, NJ 07703

Robert Thibodeau
GRC
307 Wynn Dr., N.W.
Huntsville, ALA. 35805

Robert E. Thurber
USAETL
ATTN: ETL-GS-A (Mr. Thurber)
Ft. Belvoir, VA 22060

Pierce Tolson
CSC-SGL
Ft. Lee, VA 23801

Fred T. Tracy
Waterways Experiment Station
P. O. Box 631
Vicksburg, MS 39180

LTC. W. Trueheart
USACACDA/TSM-TOS
Ft. Leavenworth, KS 66027

Lawrence A. Tubbs
BMDATC
Ballistic Missile Defense ATC
P. O. Box 1500
ATTN: BMDATC-1
Huntsvile, ALA. 35807

1LT Joseph B. Urban
USASCEFG
DTD-PMD, USASCERG
Ft. Gordon, GA. 30908

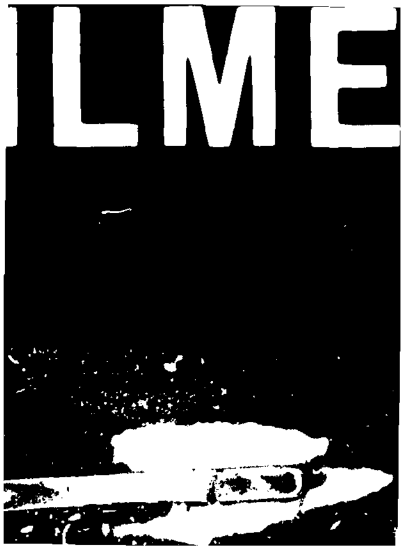
Frank E. Ward Jr.
USACORADCOM
ATTN: DRDCO-TCS-BC
Ft. Monmouth, NJ 07703

Ellwood H. Witt Jr.
USAEDPC-IS
US Army Engineer Data Processing
Center
P. O. Box 2828
Washington, D.C. 20013

Donovan Young
AIRMICS
Georgia Tech. ISYE
Atlanta, GA 30332

Jeffrey S. Yohay
USACORADCOM
ATTN: DRDCO-TCS-BG
Ft. Monmouth, NJ 07703

Saydean Zeldin
HOS
Higer Order Software, Inc.
806 Mass. Avenue
Cambridge, MA. 02139


ILME